

UNIVERSIDAD CARLOS III DE MADRID

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA



GENERADOR AUTOMÁTICO DE  
DOCUMENTACIÓN PARA EL  
SISTEMA AFABLE

AUTOR:      **ÁNGEL DE LA CALLE RABADÁN**

TUTORA:     **ANA ISABEL GONZÁLEZ-TABLAS FERRERES**

OCTUBRE 2009



**TÍTULO:**           **“Generador Automático de Documentación para el Sistema AFABLE”**

**AUTOR:**           **ÁNGEL DE LA CALLE RABADÁN**

**TUTORA:**           **ANA ISABEL GONZÁLEZ-TABLAS FERRERES**

**DIRECTORA:**   **ANA ISABEL GONZÁLEZ-TABLAS FERRERES**

**CODIRECTOR:** **RAÚL YUSTA RUÍZ**

La defensa del presente Proyecto Fin de Carrera se realizó el día \_\_\_\_ de \_\_\_\_\_ de 2009, siendo calificada por el siguiente tribunal:

**PRESIDENTE:**       \_\_\_\_\_

**SECRETARIO:**       \_\_\_\_\_

**VOCAL:**               \_\_\_\_\_

Habiendo obtenido la siguiente calificación:

**CALIFICACIÓN:**     \_\_\_\_\_

**PRESIDENTE**

**SECRETARIO**

**VOCAL**

# RESUMEN

El presente Proyecto Fin de Carrera se centra en la creación de un nuevo módulo de lógica denominado **AFABLE DOCUMENTACION**, que consiste en el diseño e implementación de una herramienta de generación automática de documentación de Planes de Abonado para Servicios 90X. Este módulo se integra, como quinto componente, dentro del Proyecto AFABLE desarrollado por el Grupo de Red Inteligente de Telefónica I + D.

La aplicación permitirá generar documentos automáticos de los Planes de Abonado totalmente personalizados, con la información de los diferentes caminos que puede seguir una llamada al Servicio 90X. Sus objetivos principales son los siguientes:

- o Obtener y clasificar los distintos itinerarios que puede seguir una llamada dentro del Plan de Abonado
- o Presentar la información relativa a cada Módulo de Lógica del Servicio 90X
- o Realizar un análisis de los Recursos del Plan de Abonado
- o Recoger toda la información generando un documento personalizado en un estándar libre de documentación

# *Índice de Contenidos*

<b>1</b>	<b>INTRODUCCIÓN .....</b>	<b>1-1</b>
1.1	CONTEXTO DE LA APLICACIÓN.....	1-1
1.2	OBJETIVOS DEL PROYECTO .....	1-3
1.3	MÉTODO DE DESARROLLO .....	1-5
1.4	ORGANIZACIÓN DE LA MEMORIA .....	1-7
<b>2</b>	<b>GESTIÓN DEL PROYECTO .....</b>	<b>2-1</b>
2.1	INTRODUCCIÓN .....	2-1
2.2	ORGANIZACIÓN DEL PROYECTO .....	2-1
2.2.1	Ciclo de vida del proyecto .....	2-1
2.2.2	Fases del proyecto.....	2-3
2.3	PROCESO DE GESTIÓN DEL PROYECTO .....	2-5
2.3.1	Gestión de riesgos .....	2-6
2.3.2	Estrategias de control de riesgos.....	2-9
2.4	PROCESO TÉCNICO .....	2-10
2.5	PLANIFICACIÓN .....	2-12
2.5.1	Planificación inicial.....	2-12
2.5.2	Planificación final .....	2-13
2.6	PRESUPUESTO .....	2-15
<b>3</b>	<b>PLANTEAMIENTO DEL PROBLEMA Y ESTUDIO DE LAS TECNOLOGÍAS .....</b>	<b>3-1</b>
3.1	ÁMBITO DEL PROYECTO.....	3-1
3.2	REQUISITOS .....	3-4
3.2.1	Requisitos funcionales .....	3-4
3.2.2	Requisitos no funcionales .....	3-8
3.3	ESTUDIO DE LAS TECNOLOGÍAS.....	3-8
3.3.1	Posibles soluciones .....	3-8
3.3.2	Soluciones adoptadas.....	3-13
3.3.2.1	Lenguaje de programación JAVA .....	3-13
3.3.2.2	Plataforma de desarrollo ECLIPSE .....	3-13

## *Índice de Contenidos (cont.)*

3.3.2.3	Estándar de documentación abierto OpenDocument (ODF) ...	3-15
---------	--	------

<b>4</b>	<b>ANÁLISIS DEL SISTEMA .....</b>	<b>4-1</b>
4.1	INTRODUCCIÓN .....	4-1
4.2	PLAN DE ABONADO VOICEXML .....	4-1
4.2.1	Grafo del Plan de Abonado VoiceXML .....	4-2
4.2.2	Módulos de Lógica (ML) .....	4-2
4.2.3	Transiciones entre Módulos de Lógica .....	4-2
4.3	DESCRIPCIÓN DE LOS MÓDULOS DE LÓGICA .....	4-3
4.3.1	NODO INICIO .....	4-8
4.3.2	NODO FIN.....	4-9
4.3.3	NODO DE CONDICIÓN IF .....	4-9
4.3.4	NODO SWITCH .....	4-10
4.3.5	NODO RUTINA ECMAScript .....	4-10
4.3.6	NODO ASIGNAR VARIABLE .....	4-11
4.3.7	NODO LOCUCIÓN .....	4-12
4.3.8	NODO MENÚ.....	4-12
4.3.9	NODO DATO ENTRADA .....	4-13
4.3.10	NODO DE GRABACIÓN DE MENSAJES .....	4-14
4.3.11	NODO TRANSFERIR LLAMADA .....	4-14
4.3.12	NODO ENVIAR CORREO .....	4-15
4.3.13	NODO ENVIAR MENSAJE AL CAR .....	4-16
4.3.14	NODO ENVIAR SMS .....	4-16
4.3.15	NODO ENVIAR MMS .....	4-17
4.3.16	NODO GEOLOCALIZAR TELÉFONO.....	4-18
4.3.17	NODO CONSULTAR BASE DE DATOS.....	4-18
4.3.18	NODO MODIFICAR BASE DE DATOS .....	4-19
4.3.19	NODO CREAR GRAMÁTICA .....	4-19
4.4	CASOS DE USO .....	4-20
4.4.1	Casos de uso de primer nivel .....	4-20
4.4.1.1	Actores de primer nivel .....	4-21
4.4.2	Casos de uso de segundo nivel.....	4-22
4.4.2.1	Actores de segundo nivel .....	4-24
4.5	DIAGRAMAS DE SECUENCIA.....	4-25
4.5.1	Diagrama de secuencia de primer nivel .....	4-25
4.5.2	Diagrama de secuencia de segundo nivel .....	4-27

## Índice de Contenidos (cont.)

<b>5</b>	<b>DISEÑO DEL SISTEMA</b>	<b>5-1</b>
5.1	INTRODUCCIÓN	5-1
5.2	LÓGICA DE LA APLICACIÓN	5-1
5.2.1	Paquete <i>(default-package)</i>	5-3
5.2.2	Paquete <i>afable.documentacion.generador</i>	5-3
5.2.3	Paquete <i>documentacion</i>	5-4
5.2.4	Paquete <i>documentacion.exceptions</i>	5-8
5.2.5	Paquete <i>documentacion.logicas</i>	5-8
5.2.6	Paquete <i>documentacion.planes</i>	5-10
5.2.7	Paquete <i>documentacion.resultados</i>	5-11
5.2.8	Paquete <i>documentacion.utils</i>	5-12
5.2.9	Paquete <i>gui</i>	5-14
5.2.10	Paquete <i>gui.interfaz</i>	5-14
5.2.11	Paquete <i>gui.utils</i>	5-15
5.2.12	Paquete <i>odf.plantilla</i>	5-16
5.2.13	Paquete <i>odf.plantilla.beans</i>	5-18
5.3	FASES DE DISEÑO	5-20
5.3.1	FASE I: Mapeo del fichero <i>.axml</i> a estructuras de datos	5-22
5.3.2	FASE II: Algoritmo de Ordenación de Nodos	5-24
5.3.3	FASE III: Clasificación y ordenación de itinerarios	5-29
5.3.4	FASE IV: Tratamiento de los recursos	5-32
5.3.5	FASE V: Lanzar interfaz gráfico y obtener respuestas del usuario	5-33
5.3.6	FASE VI: Procesado de datos para la generación del Documento Automático del Plan	5-34
5.3.7	FASE VII: Generar listados de referencia para todos los itinerarios	5-36
5.3.8	FASE VIII: Diseñar los estilos del Documento Automático del Plan	5-38
5.3.9	FASE IX: Diseñar el contenido del Documento Automático del Plan	5-42
<b>6</b>	<b>IMPLEMENTACIÓN Y PRUEBAS</b>	<b>6-1</b>
6.1	INTRODUCCIÓN	6-1
6.2	CODIFICACIÓN DEL SISTEMA	6-1
6.2.1	Documentos personalizados. Interfaz Gráfica de Usuario	6-2
6.3	IMPLEMENTACIÓN DEL DOCUMENTO ODF	6-4
6.3.1	Añadir una imagen al paquete <i>.odt</i>	6-4
6.3.2	Añadir elementos al documento <i>.odt</i>	6-5

## *Índice de Contenidos (cont.)*

6.3.2.1	Insertar estilo. Objeto OdfStyle .....	6-7
6.3.2.2	Insertar cabecera. Objeto OdfHeading .....	6-8
6.3.2.3	Insertar párrafo. Objeto OdfParagraph .....	6-9
6.3.2.4	Insertar salto de página. Objeto OdfPageBreak .....	6-10
6.3.2.5	Insertar lista. Objeto OdfList .....	6-10
6.3.2.6	Insertar tabla. Objeto OdfTable .....	6-12
6.3.2.7	Insertar una imagen. Objeto OdfFrame + OdfImage .....	6-13
6.4	ESTRUCTURA DEL DOCUMENTO AUTOMÁTICO .....	6-14
6.4.1	Portada del documento .....	6-14
6.4.2	Índice del documento .....	6-16
6.4.3	Datos del Plan .....	6-18
6.4.4	Grafo completo del Plan .....	6-21
6.4.5	Itinerario principal del Plan .....	6-22
6.4.6	Análisis de los itinerarios correctos del Plan .....	6-22
6.4.7	Análisis de los itinerarios sin salida válida del Plan .....	6-23
6.4.8	Análisis de los itinerarios con errores del Plan .....	6-24
6.4.9	Análisis de los itinerarios con eventos de marcha atrás en el Plan .....	6-25
6.4.10	Estudio de los Recursos del Plan .....	6-27
6.4.11	ANEXO: Interfaz de Usuario .....	6-32
6.5	CONTROL DE ERRORES Y EXCEPCIONES .....	6-34
6.6	SISTEMA DE TRAZAS .....	6-34
6.6.1	Niveles de prioridad de los mensajes .....	6-35
6.6.2	Appenders .....	6-35
6.6.3	Layouts .....	6-36
6.6.4	Fichero log4j_DOCUMENTACION.properties .....	6-37
6.6.5	Forma de uso de Log4j .....	6-39
6.7	PAQUETE DE LA APLICACIÓN .....	6-40
6.7.1	Fichero build.xml .....	6-40
6.7.2	AFABLE_DOCUMENTACION.zip .....	6-42
6.8	INTEGRACIÓN EN EL AFABLE IDE .....	6-44
6.9	PRUEBAS DEL SISTEMA .....	6-46
<b>7</b>	<b>CONCLUSIONES Y LÍNEAS FUTURAS DE DESARROLLO .....</b>	<b>7-1</b>
7.1	CONCLUSIONES GENERALES .....	7-1
7.2	FUTURAS MEJORAS Y ANÁLISIS CRÍTICO .....	7-4



## *Índice de Contenidos (cont.)*

<b>A</b>	<b>DESCRIPCIÓN GENERAL DEL SISTEMA AFABLE</b> .....	A-1
A.1	OBJETIVO .....	A-1
A.2	COMPONENTES DEL SISTEMA AFABLE .....	A-2
A.2.1	AFABLE IDE .....	A-4
A.2.2	AFABLE MANAGER .....	A-7
A.2.3	AFABLE SERVER .....	A-8
A.2.4	AFABLE USER WEB .....	A-9
A.3	EJEMPLO: “Servicio de consulta de notas” .....	A-9
A.3.1	Documento XML del Plan de Abonado .....	A-10
<b>B</b>	<b>MANUAL DE USUARIO</b> .....	B-1
B.1	ENTRADAS DE LA APLICACIÓN .....	B-1
B.2	INTERFAZ GRÁFICA DE USUARIO .....	B-2
	<b>GLOSARIO DE TÉRMINOS Y ACRÓNIMOS</b> .....	Glosario-1
	<b>BIBLIOGRAFÍA</b> .....	Bibliografía-1



# *Índice de Figuras*

<b>Figura 1-1:</b>	<b>Plan de Abonado “Consulta de notas” .....</b>	<b>1-1</b>
<b>Figura 2-1:</b>	<b>Ciclo de vida del proyecto .....</b>	<b>2-3</b>
<b>Figura 2-2:</b>	<b>Planificación inicial a alto nivel .....</b>	<b>2-12</b>
<b>Figura 2-3:</b>	<b>Planificación inicial de tareas a bajo nivel. Parte 1 .....</b>	<b>2-13</b>
<b>Figura 2-4:</b>	<b>Planificación inicial de tareas a bajo nivel. Parte 2 .....</b>	<b>2-13</b>
<b>Figura 2-5:</b>	<b>Planificación final de tareas a alto nivel .....</b>	<b>2-14</b>
<b>Figura 2-6:</b>	<b>Planificación final a bajo nivel. Parte 1 .....</b>	<b>2-14</b>
<b>Figura 2-7:</b>	<b>Planificación final a bajo nivel. Parte 2 .....</b>	<b>2-15</b>
<b>Figura 3-1:</b>	<b>Plan de Abonado “Elegir opción” .....</b>	<b>3-1</b>
<b>Figura 3-2:</b>	<b>Módulos Lógica Sistema AFABLE .....</b>	<b>3-2</b>
<b>Figura 3-3:</b>	<b>Integración del AFABLE DOCUMENTACIÓN dentro del Sistema AFABLE .....</b>	<b>3-4</b>
<b>Figura 3-4:</b>	<b>Vista genera de Eclipse 3.2 .....</b>	<b>3-14</b>
<b>Figura 3-5:</b>	<b>Pluggin eUML2 para Eclipse .....</b>	<b>3-15</b>
<b>Figura 3-6:</b>	<b>Tipos de ficheros soportados .....</b>	<b>3-16</b>
<b>Figura 3-7:</b>	<b>Contenedor ZIP de un fichero OpenDocument .....</b>	<b>3-16</b>
<b>Figura 3-8:</b>	<b>Fichero content.xml .....</b>	<b>3-17</b>
<b>Figura 3-9:</b>	<b>Fichero meta.xml .....</b>	<b>3-18</b>
<b>Figura 3-10:</b>	<b>Ejemplo de código de una imagen .....</b>	<b>3-18</b>
<b>Figura 3-11:</b>	<b>Modelo de capas del API ODFDOM .....</b>	<b>3-21</b>
<b>Figura 3-12:</b>	<b>Ficheros de la capa física .....</b>	<b>3-22</b>
<b>Figura 3-13:</b>	<b>Ejemplo código XML de una tabla en el documento .....</b>	<b>3-23</b>
<b>Figura 3-14:</b>	<b>Ejemplo estructura de clases de una tabla en el documento .....</b>	<b>3-23</b>
<b>Figura 4-1:</b>	<b>Nodos de Interacción con el Usuario .....</b>	<b>4-4</b>
<b>Figura 4-2:</b>	<b>Nodos de Telefonía .....</b>	<b>4-4</b>
<b>Figura 4-3:</b>	<b>Nodos básicos de Programación .....</b>	<b>4-5</b>
<b>Figura 4-4:</b>	<b>Nodos basados en Lógicas Especiales .....</b>	<b>4-6</b>
<b>Figura 4-5:</b>	<b>Ejemplo de transiciones en un Plan de Abonado .....</b>	<b>4-8</b>
<b>Figura 4-6:</b>	<b>Caso de uso de primer nivel .....</b>	<b>4-21</b>
<b>Figura 4-7:</b>	<b>Caso de uso de segundo nivel (GENERAR DOCUMENTACIÓN) .....</b>	<b>4-23</b>

## *Índice de Figuras (Cont.)*

Figura 4-8:	Diagrama de secuencia de primer nivel .....	4-27
Figura 4-9:	Diagrama de secuencia de segundo nivel .....	4-28
Figura 5-1:	Estructura lógica de paquetes .....	5-2
Figura 5-2:	Clases paquete <i>(default-package)</i> .....	5-3
Figura 5-3:	Clases paquete <i>afable.documentacion.generador</i> .....	5-4
Figura 5-4:	Clases paquete <i>documentacion</i> .....	5-7
Figura 5-5:	Clases paquete <i>documentacion.exceptions</i> .....	5-8
Figura 5-6:	Clases paquete <i>documentacion.logicas</i> .....	5-10
Figura 5-7:	Clases paquete <i>documentacion.planes</i> .....	5-10
Figura 5-8:	Clases paquete <i>documentacion.resultados</i> .....	5-12
Figura 5-9:	Clases paquete <i>documentacion.utils</i> .....	5-13
Figura 5-10:	Clases <i>paquete gui</i> .....	5-14
Figura 5-11:	Clases <i>paquete gui.interfaz</i> .....	5-15
Figura 5-12:	Clases paquete <i>gui.utils</i> .....	5-16
Figura 5-13:	Clases paquete <i>odf.plantilla</i> .....	5-18
Figura 5-14:	Clases paquete <i>odf.plantilla.beans</i> .....	5-19
Figura 5-15:	Plan de Abonado " <i>consultaHoroscopo.axml</i> " .....	5-21
Figura 5-16:	Mapeo del fichero <i>.axml</i> a estructuras de datos .....	5-23
Figura 5-17:	Algoritmo de Ordenación de Nodos. Mapeo de objetos .....	5-25
Figura 5-18:	Árbol del Plan <i>consultaHoroscopo.axml</i> .....	5-28
Figura 5-19:	Mapeo de caminos al vector de itinerarios .....	5-29
Figura 5-20:	Clasificación de los Recursos del Plan .....	5-33
Figura 5-21:	Procesado de datos del objeto <i>GrafoProcesado</i> .....	5-36
Figura 5-22:	Listado de referencia ordenado por longitud .....	5-37
Figura 5-23:	Código de estilos en el fichero <i>sytes.xml</i> .....	5-39
Figura 5-24:	Código del contenido en el fichero <i>content.xml</i> .....	5-45
Figura 6-1:	Interfaz Gráfica. Documentos personalizados .....	6-3
Figura 6-2:	Añadir imágenes al paquete <i>.odt</i> .....	6-5
Figura 6-3:	Añadir elementos al documento <i>.odt</i> .....	6-6
Figura 6-4:	Portada del Documento Automático .....	6-15

## *Índice de Figuras (Cont.)*

<b>Figura 6-5:</b>	<b>Índice del documento .....</b>	<b>6-17</b>
<b>Figura 6-6:</b>	<b>Datos del Plan de Abonado .....</b>	<b>6-20</b>
<b>Figura 6-7:</b>	<b>Grafo completo del Plan de Abonado .....</b>	<b>6-21</b>
<b>Figura 6-8:</b>	<b>Tabla de nodos del itinerario principal .....</b>	<b>6-22</b>
<b>Figura 6-9:</b>	<b>Ejemplo información Nodo de tipo Locución.....</b>	<b>6-23</b>
<b>Figura 6-10:</b>	<b>Ejemplo itinerario sin salida válida del Plan .....</b>	<b>6-24</b>
<b>Figura 6-11:</b>	<b>Ejemplo de itinerario con errores del Plan .....</b>	<b>6-25</b>
<b>Figura 6-12:</b>	<b>Ejemplo itinerario con eventos de marcha atrás en el Plan .....</b>	<b>6-26</b>
<b>Figura 6-13:</b>	<b>Ejemplo Recurso tipo Base de Datos .....</b>	<b>6-27</b>
<b>Figura 6-14:</b>	<b>Ejemplo Recurso tipo Servicio Externo. PETICIÓN .....</b>	<b>6-28</b>
<b>Figura 6-15:</b>	<b>Ejemplo Recurso tipo Servicio Externo. RESPUESTA .....</b>	<b>6-29</b>
<b>Figura 6-16:</b>	<b>Ejemplo de código xml de Respuesta de un Servicio Externo .....</b>	<b>6-30</b>
<b>Figura 6-17:</b>	<b>Ejemplo Recurso tipo Gramática .....</b>	<b>6-32</b>
<b>Figura 6-18:</b>	<b>Anexo Interfaz de Usuario .....</b>	<b>6-33</b>
<b>Figura 6-19:</b>	<b>Integración en el AFABLE IDE .....</b>	<b>6-45</b>
<b>Figura A-1:</b>	<b>Descripción general del Sistema AFABLE .....</b>	<b>A-3</b>
<b>Figura A-2:</b>	<b>Usuarios del Sistema AFABLE.....</b>	<b>A-4</b>
<b>Figura A-3:</b>	<b>Ventana principal de AFABLE IDE .....</b>	<b>A-6</b>
<b>Figura A-4:</b>	<b>Grafo del servicio de Consulta de Notas .....</b>	<b>A-10</b>
<b>Figura A-5:</b>	<b>XML Schema: PlanAbonadoVoiceXML.....</b>	<b>A-11</b>
<b>Figura B-1:</b>	<b>Pantalla de Bienvenida del Sistema de Documentación de AFABLE .....</b>	<b>B-1</b>
<b>Figura B-2:</b>	<b>Pantalla de Selección de Ficheros .....</b>	<b>B-2</b>
<b>Figura B-3:</b>	<b>Interfaz Gráfica de Usuario .....</b>	<b>B-3</b>
<b>Figura B-4:</b>	<b>Número de despliegue no válido .....</b>	<b>B-4</b>
<b>Figura B-5:</b>	<b>Tipo de ordenación inexistente .....</b>	<b>B-5</b>
<b>Figura B-6:</b>	<b>Ver todos los itinerarios del Plan de Abonado .....</b>	<b>B-6</b>
<b>Figura B-7:</b>	<b>Itinerarios individuales no válidos .....</b>	<b>B-8</b>
<b>Figura B-8:</b>	<b>Itinerarios individuales correctos .....</b>	<b>B-9</b>
<b>Figura B-9:</b>	<b>Itinerarios individuales deshabilitados .....</b>	<b>B-10</b>
<b>Figura B-10:</b>	<b>Generación correcta de la documentación.....</b>	<b>B-10</b>

## *Índice de Figuras (Cont.)*

<b>Figura B-11:</b>	<b>Error en la generación de la documentación .....</b>	<b>B-11</b>
---------------------	---	-------------

# Índice de Tablas

Tabla 2-1:	Análisis de riesgos del proyecto .....	2-8
Tabla 2-2:	Estrategias para mitigar los riesgos .....	2-9
Tabla 2-3:	Recursos humanos. Gastos de implementación .....	2-16
Tabla 2-4:	Recursos hardware. Gastos de adquisición .....	2-16
Tabla 2-5:	Elementos fungibles y costes indirectos .....	2-16
Tabla 2-6:	Presupuesto final .....	2-17
Tabla 3-1:	Estándar abierto vs Estándar cerrado .....	3-11
Tabla 3-2:	Ventajas del OpenDocument (ODF).....	3-24
Tabla 5-1:	Vector <i>posicionYNombre</i> del Plan <i>consultaHoroscopo.xml</i> .....	5-23
Tabla 5-2:	Matriz de Saltos para el Plan <i>consultaHoroscopo.xml</i> .....	5-26
Tabla 5-3:	Tabla de pesos de un nodo .....	5-31
Tabla 5-4:	Tabla de estilos del Documento Automático del Plan .....	5-40





# 1 INTRODUCCIÓN

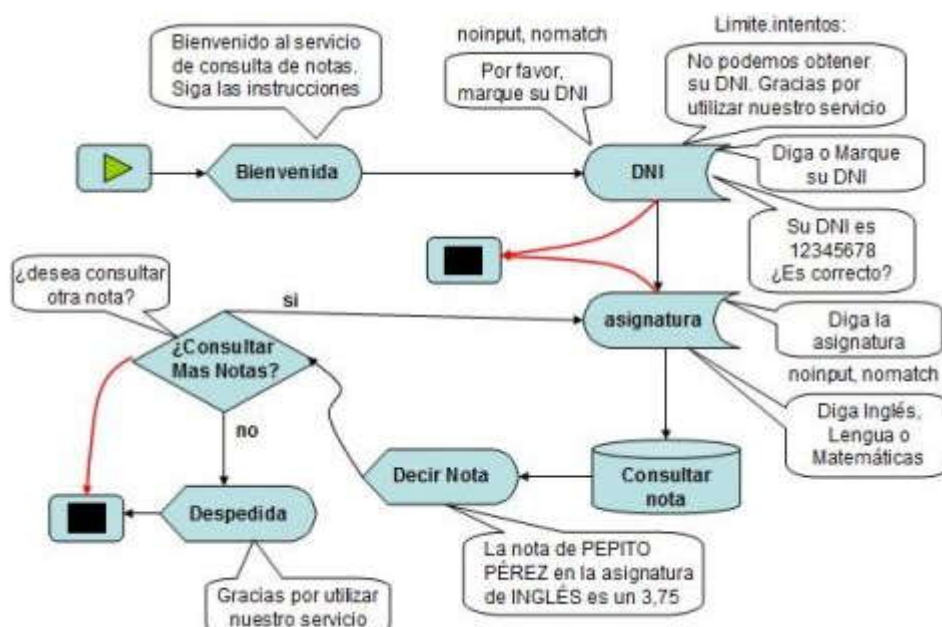
## 1.1 CONTEXTO DE LA APLICACIÓN

El Proyecto Fin de Carrera (PFC), denominado “**Generador Automático de Documentación para el Sistema AFABLE**” consistirá en diseñar e implementar un módulo de Documentación Automática de Planes de Abonado para Servicios 90X (*AFABLE DOCUMENTACIÓN*), que se integre dentro del Proyecto **AFABLE**, llevado a cabo por el Grupo de desarrollo de Red Inteligente de Telefónica I + D.

Se entiende por Plan de Abonado la descripción formal del comportamiento de un Servicio 90X. Un ejemplo de este tipo de servicios, desarrollado en el ANEXO A.3

**EJEMPLO: “Servicio de consulta de notas”** podría ser el siguiente: “*Una Universidad decide ofrecer la consulta de notas a través de un número 90X*”. El Plan de Abonado asociado al servicio tendría el siguiente aspecto:

**Figura 1-1: Plan de Abonado “Consulta de notas”**



Los Servicios 90X de Red Inteligente son servicios automáticos de atención telefónica que permiten a las empresas y negocios que los contratan una gestión rápida, económica y eficaz de todas sus llamadas. Todo ello, mediante el encaminamiento de las mismas a servicios de información y noticias, otros números de teléfono, buzones u operadora, o la realización de funciones especiales tales como el envío de SMS, MMS, mensajes al CAR y emails entre otras, en función de las características concretas de cada llamada (origen geográfico, fecha y hora de la llamada) o de la información introducida por el usuario (mediante tonos DTMF o voz) guiado por locuciones que explican las instrucciones necesarias para el uso del servicio.

El Sistema **AFABLE** (**A**utomatización de **F**unciones de **A**tención telefónica **B**asado en **L**ógicas **E**speciales) consiste en un conjunto de herramientas para dar soporte a la creación, administración y provisión de **Aplicaciones VoiceXML** asociadas a servicios 90X de Red Inteligente, con el objetivo de permitir el diseño rápido y económico de servicios 90X personalizados sin necesidad de realizar complejos desarrollos software a medida. Su característica principal es que genera servicios en formato estándar VoiceXML a partir de un diseño gráfico basado en módulos de lógica avanzados que permiten una integración sencilla y a la vez potente con los sistemas externos en los que se realizan los procesos de negocio tradicionales y de creación de contenidos.

El Sistema AFABLE está formado por cuatro componentes principales que contribuyen a la funcionalidad deseada:

- **AFABLE IDE**: Permite el diseño gráfico de un Plan de Abonado VoiceXML funcionalmente correcto.
- **AFABLE MANAGER**: Permite el despliegue y repliegue de los Planes de Abonado VoiceXML en los sistemas de Red Inteligente. Proporciona también

funciones de apoyo a la explotación tales como la consulta de informes, trazas y estado de los servicios.

- **AFABLE SERVER:** Sistema de producción que sirve las páginas VoiceXML al nodo especializado de RI para proporcionar los servicios 90X.
- **AFABLE USER WEB:** Proporciona funciones de autogestión de los servicios 90X a los titulares de los mismos a través de la Web. Por ejemplo, consulta de informes y carga masiva de datos del servicio.

El presente Proyecto Fin de Carrera se encargará del diseño e implementación de un quinto módulo de lógica:

- **AFABLE DOCUMENTACIÓN:** Proporciona de forma automática documentación personalizada del Plan de Abonado según las necesidades del titular del Servicio 90X.

## 1.2 OBJETIVOS DEL PROYECTO

En un desarrollo tradicional, un nuevo servicio debe ir acompañado de una documentación relacionada suficiente. El cometido de este Proyecto Fin de Carrera es elaborar una aplicación que reciba como entrada un plan válido, desarrollado mediante AFABLE, y que a partir del mismo, genere uno o varios documentos en un formato estandarizado.

Los objetivos concretos del Proyecto Fin de Carrera se resumen a continuación:

1. Implementar un ***algoritmo de ordenación*** que permita establecer los distintos itinerarios de un Plan de Abonado. Por **itinerario**, se entiende, el curso que sigue una llamada al Servicio 90X entre los diferentes Módulos de Lógica. Dicho algoritmo, deberá tener especial cuidado con la presencia de “bucles” dentro del Plan.

2. Clasificar los distintos itinerarios del Plan de Abonado en base al tipo de transiciones que se producen entre los distintos Módulos de Lógica. Así, se agruparán los itinerarios del Servicio 90X en cuatro grupos:
  - **Itinerarios correctos del plan.** Por itinerarios correctos se entienden aquellos caminos que toma la llamada en el Plan de Abonado en los que las transiciones entre los distintos nodos se producen de forma normal hasta la finalización de la misma.
  - **Itinerarios sin salida válida del plan.** Son aquellos en los que se produce un tipo de evento especial, por ejemplo, cuando se espera la entrada de un dato por parte del usuario y bien, pasado un tiempo, no se produce dicha interacción o bien el reconocedor de voz no puede distinguir lo que ha dicho el usuario.
  - **Itinerarios con errores.** Son aquellos en los que debido a un error de cualquier tipo la llamada no finaliza de forma normal su ejecución.
  - **Itinerarios con eventos de marcha atrás.** Aquellos en los que la llamada vuelve atrás en el plan, volviendo a pasar por nodos que ya ha atravesado.
3. Presentar la información relativa a cada Módulo de Lógica, nombre, tipo, carácter de la siguiente transición, etc., así como aquella específica de la posición que ocupa cada nodo dentro de cada itinerario del Plan de Abonado.
4. Realizar un análisis de los Recursos del Plan de Abonado. El Sistema AFABLE contempla cuatro tipos de recursos:
  - Recursos de tipo **Bases de Datos**. Son aquellos que definen una Base de Datos para poder utilizarla en el Plan de Abonado.
  - Recursos de tipo **Servicio Externo**. Aquellos que establecen la forma en la que el Sistema AFABLE puede invocar mediante una url, un servicio externo a un tercero y recibir la información mediante un formato XML concreto.
  - Recursos de tipo **Fichero**. Aquellos en los que se incorporan al plan ficheros externos de tipo genérico, como pueden ser, entre otros:

- Locuciones (ficheros de audio de extensión .mp3, .wav, .vox)
  - Gramáticas (ficheros de extensión .gsl con la definición de gramáticas)
  - Scripts (ficheros de scripts de extensión .ecs, elaborados en EcmaScript)
- Recursos de tipo **Gramática**. Son aquellos recursos que definen gramáticas para poder ser utilizadas en el Plan de Abonado.
5. Generar un documento personalizado con todos los datos mencionados anteriormente, en un formato libre. El formato elegido será “**OpenDocument Format (ODF)**” y el aspecto del mismo, desarrollado desde código Java, será similar al del resto de documentos y manuales desarrollados para los distintos Proyectos del grupo.

### 1.3 MÉTODO DE DESARROLLO

El desarrollo del presente Proyecto Fin de Carrera se puede dividir en cuatro fases claramente diferenciadas:

#### **Planteamiento del problema.**

Será la primera fase del Proyecto. En ella se abordará la carencia que presentaba el Sistema AFABLE respecto a la documentación de los Planes de Abonado y la necesidad de extender el sistema incorporando un nuevo módulo que cubriera una parte fundamental requerida insistentemente por los clientes de Servicios 90X. Se realizará una división lógica en la aplicación entre lo que es la obtención de la información del Plan de Abonado y la transformación de dicha información a un formato estandarizado.

### **Búsqueda de soluciones.**

Una vez planteado el problema, se abordarán las tecnologías necesarias para cumplir con los requisitos del sistema. Se tendrán en cuenta los siguientes aspectos:

- o Plataforma software sobre la que se ejecutará la aplicación
- o Análisis y diseño de la arquitectura de la aplicación.
- o Portabilidad y capacidad de integración de la herramienta dentro del Sistema AFABLE.
- o Estándar libre en el que se generará el documento del Plan de Abonado.
- o Tecnologías Web y Lenguajes de Programación a utilizar.

### **Análisis y diseño.**

Durante esta fase se llevará a cabo un modelo conceptual de la aplicación en base a un diagrama de clases basado en el paradigma de la Programación Orientada a Objetos (POO). Se presentarán las distintas interrelaciones existentes entre clases así como los diferentes casos de uso que se pueden dar al utilizar la aplicación.

### **Implementación y pruebas.**

Una vez superadas las fases anteriores, se probará toda la aplicación en su conjunto. Para ello, se hará uso de las distintas tecnologías planteadas en la segunda fase. Será necesario refinar el sistema hasta conseguir cumplir la especificación de requisitos del mismo. Será fundamental, en este punto, contar con una implementación de trazas suficiente, capaz de detectar los distintos problemas de forma rápida y eficaz.

## 1.4 ORGANIZACIÓN DE LA MEMORIA

La memoria del Proyecto Fin de Carrera presenta la siguiente estructura:

### **Capítulo 1. Introducción**

Se explica de forma breve el contexto en el que se desarrolla el Proyecto Fin de Carrera y el cometido principal del Sistema en el que está integrado. Se enumeran los objetivos fundamentales que pretende cubrir la aplicación, el método de desarrollo por fases que se llevará a cabo y un breve comentario acerca de la estructura de la presente memoria.

### **Capítulo 2. Gestión del Proyecto**

Contiene aspectos relacionados con el ciclo de vida del Proyecto, planificación, análisis de riesgos, metodología y herramientas utilizadas. Todo ello encaminado a la elaboración de un presupuesto final.

### **Capítulo 3. Planteamiento del problema y estudio de las tecnologías**

En este capítulo se muestra de forma clara el espacio que pretende ocupar el presente Proyecto Fin de Carrera dentro del Sistema en el que está integrado. Se analiza la especificación de requisitos que sirve de base para la construcción del Sistema y el estudio de las tecnologías que dará solución a los problemas planteados.

### **Capítulo 4. Análisis del Sistema**

En este capítulo se desarrolla de manera formal los casos de uso y diagramas de secuencia de la aplicación, además de conceptos relevantes relacionados con el ámbito de los Servicios 90X dentro de la plataforma de Red Inteligente.

## **Capítulo 5. Diseño del Sistema**

En esta sección, se presentan los diagramas lógicos de todas las clases organizadas por paquetes. Se hace una breve descripción de la funcionalidad de cada una de ellas y se analiza cada una de las fases de diseño de las que se compone la aplicación.

## **Capítulo 6. Implementación y pruebas**

Presenta, detalladamente, las particularidades de la implementación del Sistema. Se describirá la batería de pruebas efectuadas sobre ejemplos concretos de Planes de Abonado, de forma que se pueda percibir la potencia de la aplicación a la hora de generar documentación totalmente personalizada.

## **Capítulo 7. Conclusiones y líneas futuras de desarrollo**

Recoge las principales conclusiones extraídas tras la realización del presente Proyecto Fin de Carrera, así como las posibles líneas futuras de desarrollo para la aplicación y para el conjunto de tecnologías que se desarrollan.

## **ANEXO A: Descripción general del Sistema AFABLE**

Este anexo contiene la descripción general del Proyecto AFABLE, sistema en el que se integra y para el que cobra sentido el presente Proyecto Fin de Carrera.

## **ANEXO B: Manual de usuario**

Este anexo contiene el manual de usuario del Sistema.

## **Bibliografía**

Listado con las referencias utilizadas para la elaboración del Proyecto Fin de Carrera.



---

## **2 GESTIÓN DEL PROYECTO**

### **2.1 INTRODUCCIÓN**

En el presente capítulo se analizarán todos los aspectos relacionados con la Gestión del Proyecto. En primer lugar, se hará un análisis del ciclo de vida del proyecto, estableciendo el modelo de proceso desde el punto de vista de la Ingeniería del Software. Se hará una descripción de cada una de las fases que componen el modelo y se presentarán las principales ventajas del mismo. En segundo lugar, se analizará el Proceso de Gestión del Proyecto, análisis de riesgos, estrategias y mecanismos de control y monitorización para minimizarlos.

En tercer lugar, se enumerará el conjunto de herramientas hardware y software utilizados en el proyecto. Finalmente, se detallará la planificación del Proyecto y se realizará un análisis económico orientado a elaborar un presupuesto final.

### **2.2 ORGANIZACIÓN DEL PROYECTO**

#### **2.2.1 Ciclo de vida del proyecto**

Un modelo de ciclo de vida define el estado de las fases a través de las cuales se mueve un proyecto de desarrollo de software. Básicamente, es una vista de las actividades que ocurren durante el desarrollo de software que intenta determinar el orden de las etapas involucradas y los criterios de transición asociados entre sus etapas.

El ciclo de vida de un proyecto:

- Describe las fases principales de desarrollo de software.
- Define las fases primarias que se espera se ejecuten durante esas fases.

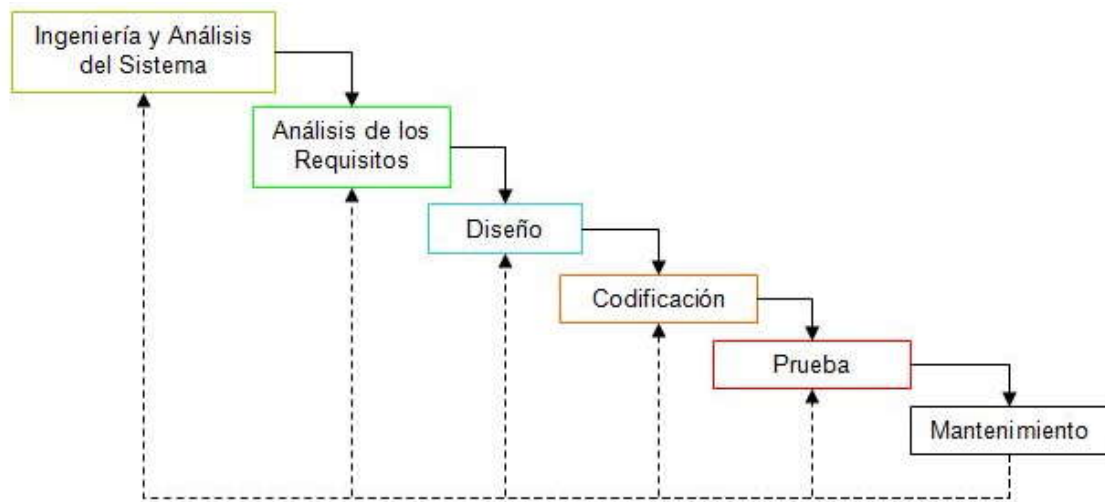
- Ayuda a administrar el progreso del desarrollo
- Provee un espacio de trabajo para la definición de un detallado proceso de desarrollo del software.

Así, los modelos, por una parte suministran una guía para los ingenieros de software con el fin de ordenar las diversas actividades técnicas en el proyecto y por otra, suministran un marco para la administración del desarrollo y el mantenimiento, en el sentido en que permiten estimar recursos, definir puntos de control intermedios, monitorizar el avance, etc.

El ciclo de vida del presente Proyecto Fin de Carrera se denomina en ingeniería del Software ***“Modelo en cascada con retroceso ó retroalimentación”***. Este ciclo de vida es lineal tal y como lo es el modelo en el que se basa, pero presenta ciertas correcciones al *“Modelo en cascada puro”*, de forma que le dotan de una menor rigidez y de una mayor versatilidad.

En un modelo en cascada, un proyecto progresa a través de una secuencia ordenada de pasos que van desde una especificación de requisitos hasta una fase de mantenimiento. El método realiza una revisión al final de cada etapa para determinar si se puede continuar hacia la siguiente fase. Cuando la revisión determina que el proyecto no está listo para pasar a la siguiente etapa, permanece en la fase actual hasta que esté preparado. Las ***“retroalimentaciones”*** permiten, que en caso de fallo, se pueda volver siempre a cualquier fase anterior para corregir el problema. De esta manera, no nos cerramos en un ciclo clásico evitando en la medida de lo posible los bloqueos en el desarrollo de la aplicación.

El modelo descrito se representa en la siguiente figura:

**Figura 2-1: Ciclo de vida del proyecto**

### **Ventajas**

- Ayuda a localizar errores en las primeras etapas del proyecto a un bajo coste.
- Ayuda a minimizar los gastos de planificación.
- Es especialmente sencillo, ya que sigue los pasos intuitivos a la hora de desarrollar el software.

### **2.2.2 Fases del proyecto**

A continuación se muestra una descripción de cada una de las fases del modelo:

#### **Ingeniería y Análisis del Sistema**

Debido a que el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software.

## **Análisis de los Requisitos**

En esta fase se elabora una especificación completa de los requisitos del Sistema, funciones requeridas, interfaces y rendimiento que se espera conseguir del software.

## **Diseño**

El diseño del software se enfoca en cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación.

## **Codificación**

El diseño debe traducirse en una forma legible para la máquina. El paso de codificación realiza esta tarea. Si el diseño se realiza de una manera detallada la codificación puede realizarse mecánicamente. En la fase de implementación se desarrolla lo estipulado en el diseño y se construye el sistema que será sometido a pruebas en la fase siguiente.

## **Prueba**

Una vez que se ha generado el código comienza la prueba del programa. La prueba se centra en la lógica interna del software, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren. En cualquier momento se puede retroceder a una fase anterior si las circunstancias lo requieren o nuevas funcionalidades son requeridas.

## **Mantenimiento**

El software puede sufrir cambios después de su entrega al cliente. Los cambios ocurrirán debidos a que se hayan encontrado errores, a que el software deba adaptarse a cambios del entorno externo (sistema operativo o dispositivos periféricos), o debidos a que el cliente requiera ampliaciones funcionales o de rendimiento.

## **2.3 PROCESO DE GESTIÓN DEL PROYECTO**

En este apartado, se va a analizar el Proceso de Gestión del Proyecto. Se plantearán los principales objetivos de la gestión del proyecto y se desarrollarán los riesgos existentes para la realización del mismo.

Un proyecto software es el conjunto de actividades técnicas y de gestión requeridas para entregar al cliente la herramienta solicitada en base a unos requisitos previamente establecidos. Tiene un tiempo de vida limitado, consume recursos y genera productos de trabajo.

El proceso de gestión está destinado a la planificación y organización de los recursos disponibles para controlar y dirigir el proyecto software.

Entre los objetivos de la gestión del proyecto software se encuentran disminuir costes, satisfacer fechas de entrega, conseguir un buen producto y prever posibles contingencias.

Los objetivos de la gestión de este proyecto son:

- Desarrollar una aplicación software que cumpla los requisitos especificados, consiguiendo una herramienta de calidad para los clientes.
- Realizar las modificaciones necesarias para integrar este quinto módulo que supone la herramienta junto con los otros cuatro módulos que componen el Sistema AFABLE.

- Aprender los aspectos teóricos y técnicos de las diferentes utilidades software usados en el desarrollo de la aplicación.
- Diseñar el sistema para su funcionamiento autónomo, de forma que se pueda ejecutar la aplicación en cualquier PC sin complejos procesos de instalación.
- Desarrollar un producto fácil de mantener y ampliable, de forma que se pueda extender su funcionalidad con nuevas mejoras en un futuro.
- Elaboración de una memoria que explique detalladamente todos los aspectos del Proyecto, así como las interconexiones con el resto de módulos del Sistema AFABLE.
- Respetar plazos de entrega.

### 2.3.1 Gestión de riesgos

La **Gestión de riesgos** es un enfoque estructurado para manejar la incertidumbre relativa a una amenaza, a través de una secuencia de actividades humanas que incluyen evaluación de riesgos, estrategias de desarrollo para manejarlos y mitigación del riesgo utilizando recursos gerenciales. Las estrategias incluyen transferir el riesgo a otra parte, evadir el riesgo, reducir los efectos negativos del riesgo y aceptar algunas o todas las consecuencias de un riesgo particular.

Un riesgo, por tanto, es una valoración subjetiva de la probabilidad de no alcanzar los objetivos deseados en términos de tiempo, coste y recursos asignados. Es la posibilidad de sufrir pérdidas durante el ciclo de vida del proyecto. En un proyecto en desarrollo, la pérdida supone una disminución de la calidad del producto final, costes más elevados, retrasos, o fallos en el proyecto.

El riesgo siempre implica dos características:

- **Incetidumbre.** El acontecimiento que caracteriza el riesgo puede o no puede ocurrir.

- **Pérdida.** Si el riesgo se convierte en una realidad, ocurrirán consecuencias no deseadas o pérdidas. Cuando se analizan los riesgos es importante cuantificar el nivel de incertidumbre y el grado de pérdidas asociado con cada riesgo.

Para el análisis de riesgos vamos a establecer tres categorías:

1. **Riesgos del proyecto:** Amenazan al plan del proyecto. Si los riesgos del proyecto se hacen realidad, es probable que la planificación temporal del proyecto se retrase y que los costes aumenten. Los riesgos del proyecto identifican los problemas potenciales de presupuesto, planificación temporal, personal y de asignación de recursos con el consecuente impacto en el proyecto software.
2. **Riesgos técnicos:** Amenazan la calidad y la planificación temporal del software que hay que producir. Si un riesgo técnico se convierte en realidad, la implementación puede llegar a ser difícil o imposible. Los riesgos técnicos identifican problemas potenciales de diseño, implementación, de interfaz y de mantenimiento. Además, la ambigüedad de especificaciones, incertidumbre técnica, técnicas anticuadas y las "tecnologías punta" son también factores de riesgo. Los riesgos técnicos ocurren porque el problema es más difícil de resolver de lo que pensábamos.
3. **Riesgos del negocio:** Amenazan la viabilidad del software a construir. Los riesgos del negocio a menudo ponen en peligro el proyecto o el producto.

Definiremos para cada riesgo en concreto la probabilidad de que ocurra, según los siguientes valores:

- **ALTA:** Probabilidad elevada de que el riesgo se convierta en realidad.
- **MEDIA:** Probabilidad razonable de que el riesgo se convierta en realidad.
- **BAJA:** Probabilidad escasa de que el riesgo se materialice.

Utilizaremos los mismos indicadores para indicar la severidad de cada tipo de riesgo:

- **ALTA**: El riesgo es muy importante y será prioritario su tratamiento.
- **MEDIA**: El riesgo es considerable pero su tratamiento tiene menor prioridad.
- **BAJA**: El riesgo es poco importante por lo que se pueden establecer tratamientos de baja prioridad.

Identificaremos, también, para cada tipo de riesgo, la etapa o etapas del desarrollo que se ven afectadas:

- **(R)**: Fase de Requisitos.
- **(D)**: Fase de Diseño.
- **(C)**: Fase de Codificación.
- **(P)**: Fase de Pruebas.
- **(M)**: Fase de Mantenimiento:
- **TODAS**: Todas las fases.

Siguiendo estas pautas el análisis de riesgos se puede observar en la siguiente tabla:

**Tabla 2-1: Análisis de riesgos del proyecto**

RIESGOS DEL PROYECTO				
Id.	Probabilidad	Severidad	Descripción	Fases
1	MEDIA	MEDIA	Riesgo en la gestión y planificación del proyecto	(R) (D)
2	ALTA	BAJA	Falta de tiempo para cumplir los plazos debido a otros compromisos laborales del desarrollador	TODAS
3	MEDIA	MEDIA	Desconocimiento del tamaño real de la aplicación	(R) (D)
4	BAJA	MEDIA	Cambio de requisitos por parte del cliente	(R) (D)
5	BAJA	ALTA	Pérdida de documentos o productos software	TODAS
6	BAJA	ALTA	Robos materiales	TODAS



7	BAJA	ALTA	Catástrofe natural	TODAS
8	BAJA	ALTA	Enfermedad	TODAS
<b>RIESGOS TÉCNICOS</b>				
<b>Id.</b>	<b>Probabilidad</b>	<b>Severidad</b>	<b>Descripción</b>	<b>Fases</b>
9	ALTA	MEDIA	Retrasos debidos a falta de formación y experiencia en las tecnologías a utilizar	(D) (C)
10	MEDIA	ALTA	Falta de soporte y documentación en algunas de las tecnologías utilizadas, debido a que aún se encuentran en desarrollo	(C) (P)
11	MEDIA	MEDIA	Ambigüedad de las especificaciones técnicas	(D) (C)
<b>RIESGOS DEL NEGOCIO</b>				
<b>Id.</b>	<b>Probabilidad</b>	<b>Severidad</b>	<b>Descripción</b>	<b>Fases</b>
12	BAJA	ALTA	Construir un sistema excelente que no quiere nadie en realidad (riesgo de mercado)	(R)
13	BAJA	MEDIA	Construir un producto que el departamento de ventas no sabe cómo vender	(R) (D)
14	BAJA	ALTA	Construir un producto que no encaja en la estrategia general de la compañía (riesgo estratégico)	(R)

### 2.3.2 Estrategias de control de riesgos

Las estrategias planeadas para mitigar los riesgos del apartado anterior se recogen en la siguiente tabla:

**Tabla 2-2: Estrategias para mitigar los riesgos**

<b>ESTRATEGIAS</b>	
<b>Id.</b>	<b>Descripción</b>
1	Reuniones con el tutor de la empresa y consultas con la tutora de la universidad, que disponen

	de más experiencia en la gestión de proyectos
2	Organización de la jornada laboral para dedicar un tiempo fijo al Proyecto, planificación en base a este tiempo.
3	Consultas con el tutor de la empresa y la tutora de la universidad
4	Plantear un diseño flexible que permita incorporar cambios en fases anteriores sin demasiado impacto en el desarrollo de la aplicación
5	Disponer de soporte de backup donde ir almacenando periódicamente copias de seguridad de la aplicación.
6	Extremar las medidas de seguridad en el entorno de trabajo. Proteger nuestro equipo de trabajo cuando no estemos trabajando en él
7	Es inevitable
8	Aunque es prácticamente inevitable, conviene vigilar la alimentación y los períodos de descanso durante la realización del proyecto
9	Labor de documentación, a través del estudio de la bibliografía, la participación en foros de debate y la consulta a personal especializado con experiencia en cada una de las tecnologías
10	Investigar el estado actual de la tecnología. Intervenir en los foros y blogs donde se debaten los problemas y prestaciones actuales y las líneas de futuro que se pretenden conseguir
11	Consultas con el tutor de la empresa y la tutora de la universidad
12	Plantear la viabilidad de la herramienta y la especificación de requisitos a los especialistas de mercado capaces de valorar su acogida entre los clientes.
13	Estar en contacto con quiénes tienen que dar salida a la herramienta en el mercado
14	Presentar la herramientas mediante demos si fueran necesarias a los jefes y a quiénes dispongan de responsabilidad para establecer si entra o no dentro de la línea estratégica de la compañía

## 2.4 PROCESO TÉCNICO

En esta sección se van a enumerar todos los elementos hardware y software que se han utilizado para la elaboración de las diferentes fases del Proyecto.

### **Elementos hardware**

- Ordenador sobremesa DELL Optiplex GX280. Pentium IV 2.8 GHz con 1 Gb de RAM. Utilizado para el desarrollo de la aplicación.
- Ordenador portátil TOSHIBA A300-228 Intel Core 2 Duo T6600 con 3 Gb de RAM. Utilizado para la memoria y para la parte final del desarrollo.
- Impresora HP Photosmart 7450. Empleada para la impresión de documentos.
- Memoria USB PNY 4Gb. Utilizada como dispositivo de backup y para demostraciones.

### **Elementos software**

- Entorno: Windows XP Professional Service Pack 3 y Windows Vista Home Premium con Service Pack 2
- IDE: Eclipse 3.2 para el grueso del proyecto y Netbeans 6.5.1 para el manejo de JSwing
- Máquina Virtual de Java: JDK 1.5
- Generación de documentación automática: OpenOffice 3.0
- Planificación: Microsoft Project 2003
- Documentación: Microsoft Word 2003, Acrobat Reader 8.1

### **Elementos fungibles**

Son básicamente el material normal de oficina (folios, bolígrafos, cartuchos de tinta, cuadernos, etc.).

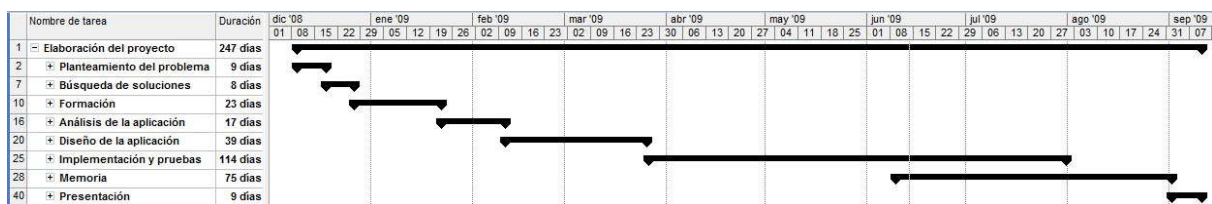
## 2.5 PLANIFICACIÓN

### 2.5.1 Planificación inicial

En la planificación inicial se ha estimado que el Proyecto ha sido realizado por una sola persona que le ha dedicado un cuarto de la jornada laboral, complementado con períodos de tiempo fuera del horario laboral hasta llegar a una media aproximada de 20 horas semanales. Para organizar la información se hará uso de un diagrama de Gantt, donde se muestra de forma gráfica las diferentes tareas programadas inicialmente como parte del proyecto.

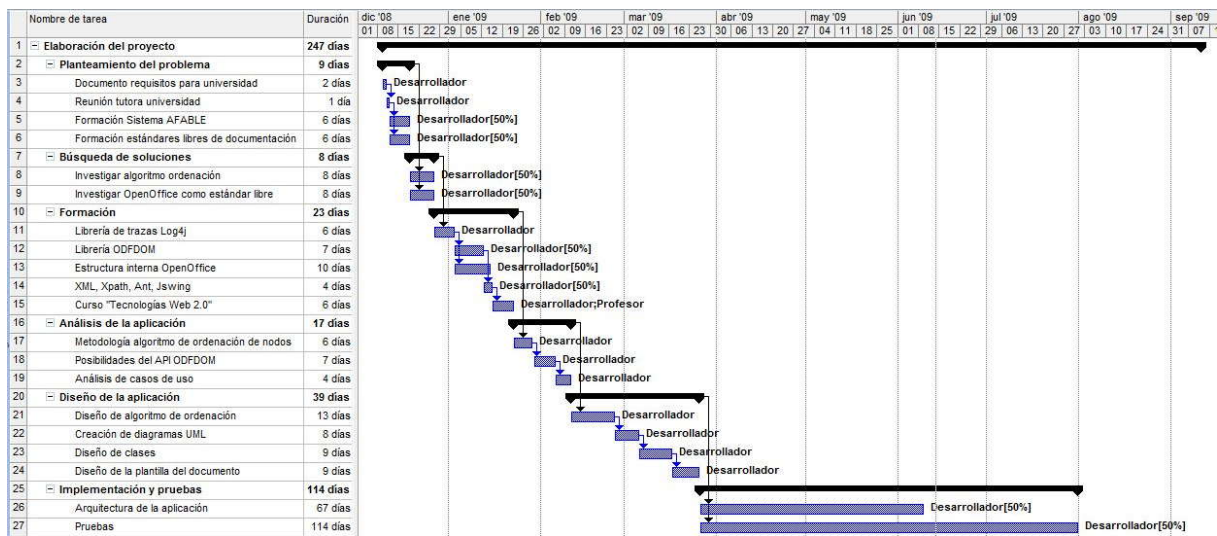
En la siguiente figura, se puede ver un desglose a alto nivel de las tareas principales del proyecto.

**Figura 2-2: Planificación inicial a alto nivel**

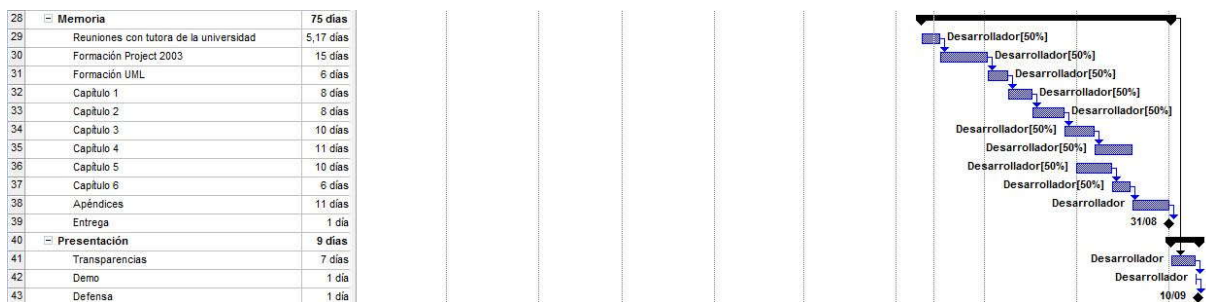


Se han dividido las tareas en ocho grupos, de los cuales, los primeros seis corresponden a la elaboración del proyecto en sí, y los dos últimos se corresponden a las tareas relacionadas con la memoria y la presentación de cara a la defensa del proyecto.

El desglose de los seis primeros grupos de tareas relacionados con el proyecto se recoge en la siguiente figura.

**Figura 2-3: Planificación inicial de tareas a bajo nivel. Parte 1**

En lo que se refiere a las tareas relacionadas con la memoria y la presentación del proyecto, el desglose a bajo nivel se muestra en la siguiente figura:

**Figura 2-4: Planificación inicial de tareas a bajo nivel. Parte 2**

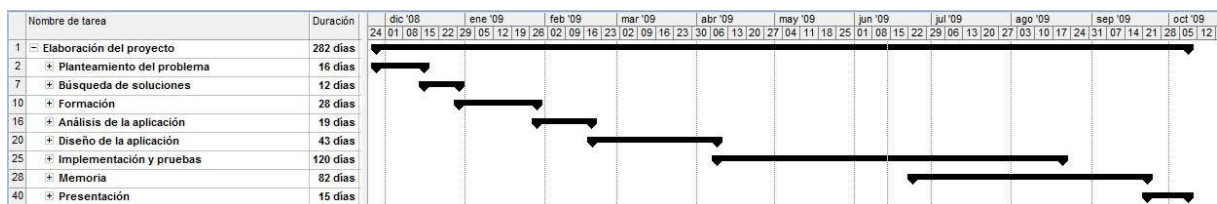
## 2.5.2 Planificación final

Una vez cubiertas las primeras fases del desarrollo, ha sido necesario modificar la planificación inicial para dar una representación real de la gestión del proyecto. Con respecto a la planificación inicial ha sido necesario ampliar los plazos, motivados por necesidades de formación, dificultades añadidas motivadas por otros aspectos

laborales y por la inclusión de ampliaciones de funcionalidad impuestas sobre la marcha.

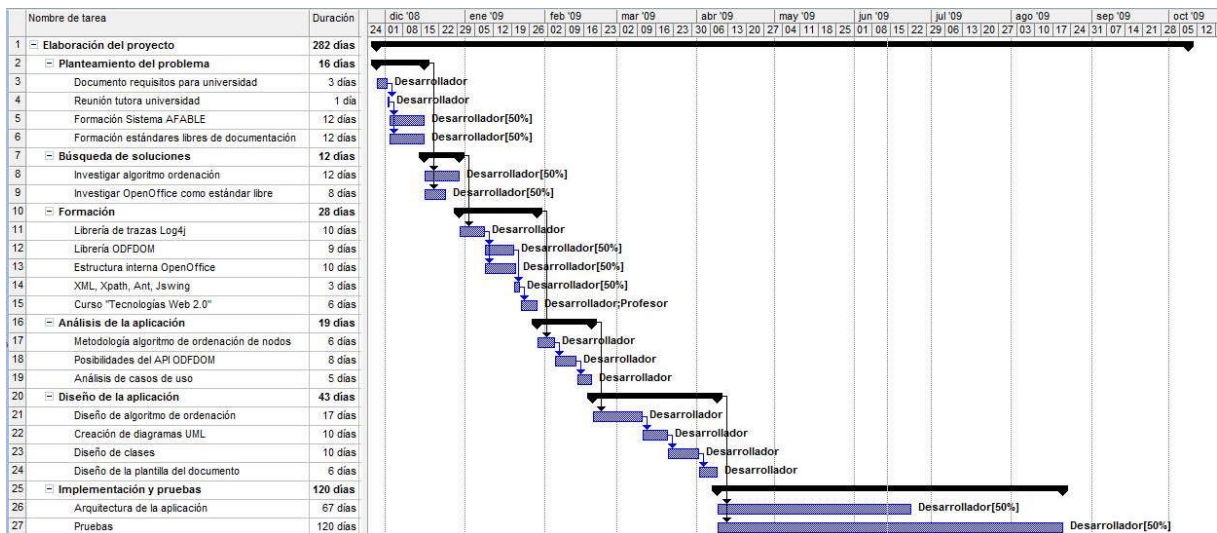
El desglose a alto nivel de los ocho grupos de tareas se muestra en la siguiente figura:

**Figura 2-5: Planificación final de tareas a alto nivel**

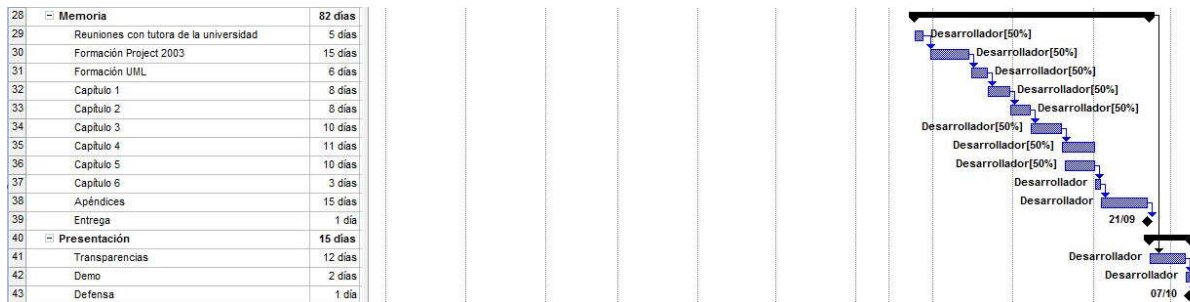


El análisis de los grupos de tareas relacionadas con el proyecto en sí, se detalla en la siguiente imagen:

**Figura 2-6: Planificación final a bajo nivel. Parte 1**



El desglose de tareas para los dos grupos relacionados con la memoria del proyecto se muestra en la siguiente figura:

**Figura 2-7: Planificación final a bajo nivel. Parte 2**

## 2.6 PRESUPUESTO

En este apartado, se va a realizar un análisis económico del Proyecto. Para la realización de un proyecto software es necesario emplear distintos tipos de recursos a fin de cuantificar la inversión prevista. A continuación se enumeran los distintos tipos de recursos utilizados para el proyecto junto con la forma en que han sido estimados y contabilizados:

- **Recursos humanos:** Se hará un cálculo de las horas empleadas por el desarrollador para la elaboración del proyecto en cada una de las fases.
- **Elementos fungibles:** Se hará una estimación del coste de los distintos materiales de oficina utilizados en el Proyecto, como pueden ser folios, cartuchos de tinta, cuadernos, bolígrafos, etc. Se han contabilizado en unidades de compra.
- **Herramientas software:** Se contabilizará el precio de las distintas licencias de software utilizadas, dentro del coste de los equipos, por lo que no computarán en el presupuesto final.
- **Herramientas hardware:** Coste de adquisición de los equipos.
- **Costes indirectos:** Como pueden ser desplazamientos, llamadas de teléfono, etc.

En la siguiente tabla se muestran los costes derivados de los recursos humanos:

**Tabla 2-3: Recursos humanos. Gastos de implementación**

FASE	DURACIÓN	COSTE	IMPORTE
Requisitos	44 horas	20 €/hora	880 €
Análisis	216 horas	18 €/hora	3.888 €
Diseño	172 horas	15 €/hora	2.580 €
Codificación	304 horas	15 €/hora	4.560 €
Pruebas	132 horas	10 €/hora	1.320 €

A continuación se muestran los costes derivados de los elementos hardware (incluyen el precio de las licencias software necesarias):

**Tabla 2-4: Recursos hardware. Gastos de adquisición**

CONCEPTO	IMPORTE
Ordenador sobremesa DELL Pentium IV	650 €
Ordenador TOSHIBA Intel Core 2 Duo	950 €
Impresora HP Photosmart 7450	110 €
Memoria USB PNY 4 Gb	12 €

En la siguiente tabla se muestra los costes derivados de los elementos fungibles y costes indirectos:

**Tabla 2-5: Elementos fungibles y costes indirectos**

CONCEPTO	IMPORTE
Elementos fungibles	120 €
Costes indirectos	60 €



A partir de estos datos el presupuesto total es el mostrado en la siguiente tabla:

**Tabla 2-6: Presupuesto final**

CONCEPTO	IMPORTE
Recursos humanos	13.228 €
Recursos hardware	1.722 €
Elementos fungibles	120 €
Costes indirectos	60 €
Base imponible	15.131 €
I.V.A (16 %)	2.420,8 €
<b>TOTAL</b>	<b>17.551,8 €</b>



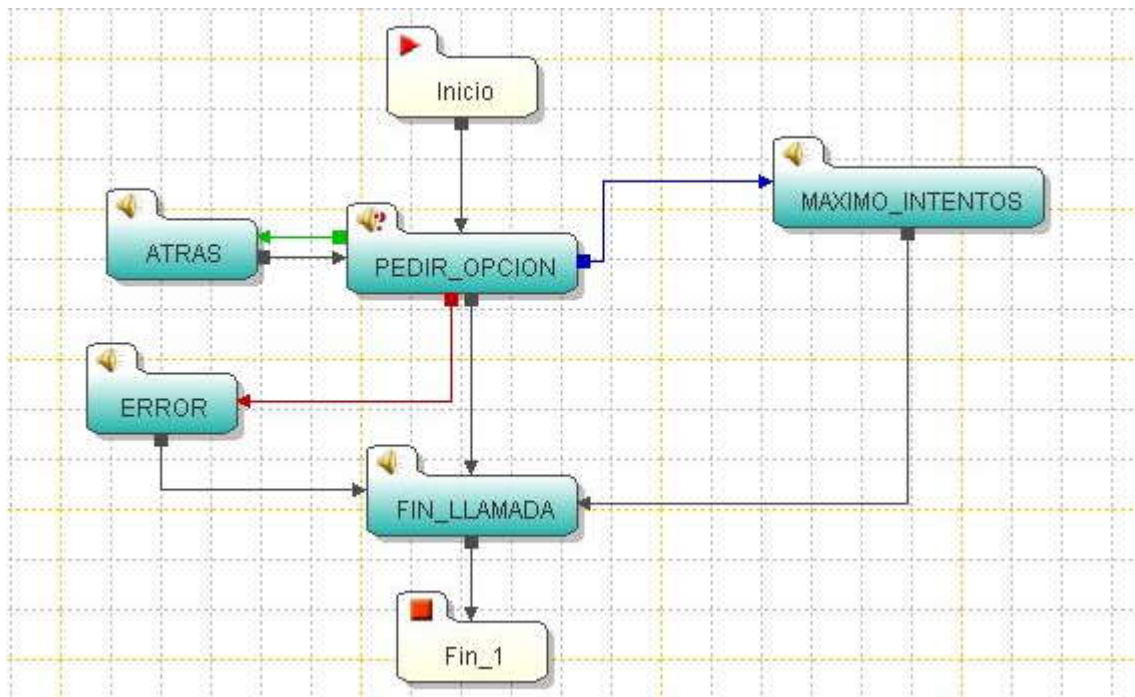
## 3 PLANTEAMIENTO DEL PROBLEMA Y ESTUDIO DE LAS TECNOLOGÍAS

### 3.1 ÁMBITO DEL PROYECTO

En este capítulo se planteará el problema que trata de solucionar este PFC, así como los requisitos funcionales y no funcionales, en base a los cuáles, se construye la aplicación.

Imaginemos un Servicio 90X muy sencillo cuyo Plan de Abonado asociado consiste únicamente en solicitar al usuario que elija una determinada opción. La representación gráfica de los Módulos de Lógica por los que se va cursando la llamada podría ser la siguiente:

**Figura 3-1: Plan de Abonado “Elegir opción”**

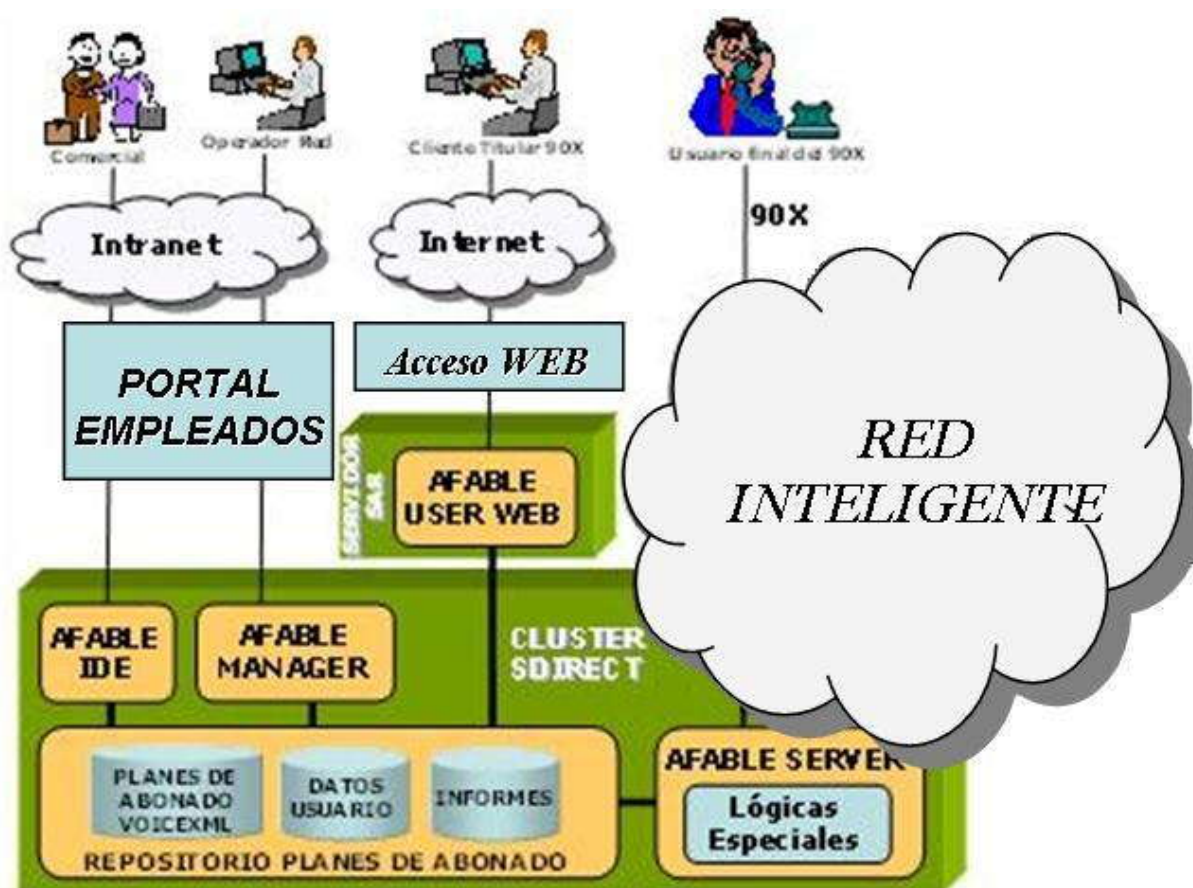


Cuando un usuario llama al número 90X, una locución le invita a introducir la opción que desee. Si el dato se procesa correctamente la llamada finaliza sin más tratamiento. Si por el contrario se produce un error a la hora de recoger el dato, o

bien el reconocedor de voz no es capaz de detectar o reconocer la opción introducida, se procede a informar de la situación al usuario mediante distintas locuciones antes de que finalice la llamada.

El papel que representa cada uno de los Módulos de Lógica que componen el Sistema AFABLE, aplicado a este Plan de Abonado, se describe a continuación:

**Figura 3-2: Módulos Lógica Sistema AFABLE**



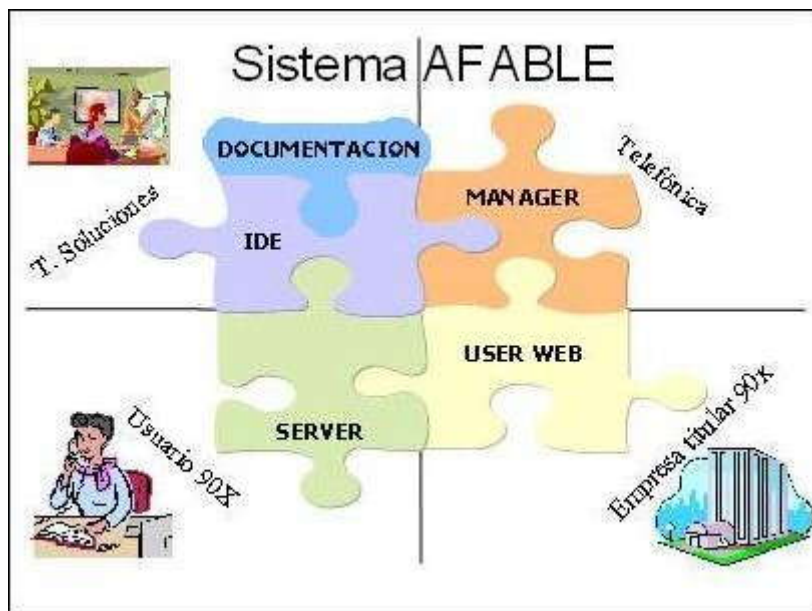
- **AFABLE IDE:** Es la interfaz gráfica que permite a los comerciales del operador diseñar el Plan de Abonado tal y como se representa en la primera figura.
- **AFABLE MANAGER:** Es la interfaz web que permite a los administradores desplegar, replegar y activar el Plan de Abonado en los nodos de Red Inteligente.

- **AFABLE SERVER:** Es el módulo especializado que permitirá el intercambio de locuciones entre la Red Inteligente y el usuario final que llama al servicio.
- **AFABLE USER WEB:** Es el módulo que permite al titular del Servicio 90X consultar por internet datos relacionados con su Plan de Abonado, como pueden ser informes de llamadas.

Desde hace ya algún tiempo, los clientes de Servicios 90X vienen demandando una documentación completa y detallada de los Planes de Abonado que conforman sus servicios. El éxito del estándar VoiceXML y la enorme gama de posibilidades que permite el Sistema AFABLE provoca que, cada vez, los Servicios 90X tengan una lógica más compleja y a la vez, más adaptada a las necesidades concretas de los clientes. La posibilidad de manejar los diferentes caminos que puede cursar una llamada a un Servicio 90X o la capacidad de unir en un documento los datos de todas las locuciones o todos los recursos de un Plan de Abonado, confieren a la herramienta que constituye el presente Proyecto Fin de Carrera, un papel protagonista a la hora de completar y comercializar el Sistema AFABLE, situándolo como claro referente en la generación de servicios de voz basados en plataformas de Red Inteligente.

El presente Proyecto Fin de Carrera, desarrollará un nuevo módulo para el Sistema AFABLE, denominado **AFABLE DOCUMENTACIÓN**, que se integrará dentro del AFABLE IDE como se muestra en la siguiente figura.

**Figura 3-3: Integración del AFABLE DOCUMENTACIÓN dentro del Sistema AFABLE**



## 3.2 REQUISITOS

A continuación se van a explicar los requisitos impuestos por el cliente para la implementación del proyecto, además de las restricciones y necesidades manifestadas por el jefe de proyecto y tutor en la empresa. Se van a clasificar los requisitos en dos grupos:

- **Requisitos funcionales:** Aquéllos que describen la funcionalidad del sistema, es decir, lo que el sistema ofrece al usuario.
- **Requisitos no funcionales:** Aquéllos que especifican, entre otras cosas, las restricciones software y hardware de la aplicación.

### 3.2.1 Requisitos funcionales

Los requisitos funcionales se recogen a continuación:

### **Requisitos referentes a la arquitectura de la aplicación**

1. La aplicación deberá funcionar de forma autónoma en cualquier PC, sin que sea necesario ningún tipo de instalación previa.
2. El Sistema deberá estar empaquetado en un fichero de archivos comprimidos, típicamente un paquete de extensión *.zip*, que una vez descomprimido se pueda ejecutar mediante un fichero de extensión *.bat*
3. Además de funcionar de forma autónoma e independiente, deberá también funcionar integrado dentro del Sistema AFABLE, para lo cuál es necesario incorporar una opción de **“Generar documentación”**, dentro del interfaz gráfico del AFABLE IDE.

### **Requisitos referentes a la implementación de la aplicación**

4. La aplicación deberá disponer de un ***algoritmo de ordenación de nodos***, que permita establecer los distintos itinerarios que puede seguir una llamada al ejecutar un Servicio 90X. Dicho algoritmo tomará como entradas los posibles siguientes saltos que posee cada uno de los Módulos de Lógica en el Plan de Abonado y presentará como salida un grupo finito de caminos compuestos por Módulos de Lógica en orden de ejecución.
5. Dicho algoritmo de ordenación deberá resolver el problema de los ciclos repetitivos dentro de un Plan de Abonado. Para ello, dispondrá de un ***sistema “antibucles”***, que permita detectar el momento en el que una llamada pasa por un Módulo de Lógica que ya ha atravesado con anterioridad. La aplicación, una vez que se detecta el bucle, deberá finalizar el itinerario, tomando el Módulo de Lógica repetido como último nodo del camino.
6. El Sistema deberá clasificar los distintos itinerarios en función del tipo de transiciones existentes entre sus Módulos de Lógica, tal y como se recoge en el apartado **1.2 OBJETIVOS DEL PROYECTO**

7. Para cada grupo de itinerarios se mostrará del mismo color que su transición, el nombre del nodo a partir del cuál, la llamada al Servicio 90X toma un camino diferente del previsto como normal dentro del Plan de Abonado.
8. Para cada itinerario concreto, será necesario presentar la información más relevante de cada uno de los Módulos de Lógica que lo componen, prestando especial atención a los nodos que incorporan locuciones al Plan de Abonado. Se deberá indicar, para aquéllos nodos en los que proceda, la opción de salida que ha tomado la llamada en ese itinerario. Por ejemplo, en un nodo de tipo *“if”*, si la transición hacia el siguiente nodo se ha producido por la rama de *CIERTO* o por la rama de *FALSO*.
9. La aplicación deberá también recoger la información de los recursos del Plan de Abonado, tal y como se describe en el apartado **1.2 OBJETIVOS DEL PROYECTO**

### **Requisitos referentes a la documentación generada**

10. La aplicación deberá generar de forma automática un documento de texto con toda la información requerida en los apartados anteriores. Dicho documento deberá estar redactado en un estándar libre de documentación.
11. El documento automático generado por la aplicación deberá estar totalmente personalizado en base a los deseos del cliente del Servicio 90X. Para ello, la aplicación deberá contar con una interfaz gráfica donde el usuario responda a una serie de preguntas y elija entre un conjunto de opciones de configuración.
12. El usuario podrá elegir un tipo de ordenación concreta para sus itinerarios, de entre las dos siguientes:
  - ***Ordenación por Longitud***. Los itinerarios se ordenan en función del número de nodos, de mayor a menor.



- ***Ordenación por Importancia.*** Los itinerarios se ordenan, de mayor a menor, en función del peso que tiene cada nodo dentro de su itinerario.
13. La aplicación deberá disponer de un ***algoritmo*** que calcule el ***peso de un itinerario*** a partir del peso particular de cada uno de los Módulos de Lógica que lo forman.
14. La aplicación deberá generar un documento automático del Plan de Abonado, que contendrá una serie de secciones concretas, en base a las opciones elegidas por el usuario en la interfaz gráfica. Las secciones del documento y su carácter, obligatorio u opcional, se enumeran a continuación:
- *Portada del documento* (obligatorio)
  - *Índice del documento* (obligatorio)
  - *Grafo completo del plan* (opcional)
  - *Datos generales del plan* (obligatorio)
  - *Itinerario principal del plan* (opcional)
  - *Itinerarios correctos del plan* (opcional)
  - *Itinerarios sin salida válida del plan* (opcional)
  - *Itinerarios con errores del plan* (opcional)
  - *Itinerarios con eventos de marcha atrás en el plan* (opcional)
  - *Recursos del plan* (obligatorio)
  - *Anexo con la interfaz gráfica de usuario* (obligatorio)
15. El documento generado por la aplicación deberá tener el mismo formato que el resto de documentos generados por el Grupo de desarrollo y todos los componentes del documento, encabezados, índices, estilos, etc., deberán ser programados desde el código fuente.

### 3.2.2 Requisitos no funcionales

Los requisitos no funcionales se detallan a continuación:

1. El sistema debe ser robusto y funcionar para cualquier tipo de Plan de Abonado.
2. El sistema debe generar documentación en un estándar de documentación libre.
3. El lenguaje de programación a utilizar para implementar el sistema será Java 1.5. Deberá comprobarse la compatibilidad de la utilización de este lenguaje con el resto de requisitos de diseño tecnológicos.
4. La herramienta será una aplicación Java “*stand-alone*”, con el fin de que se pueda utilizar en cualquier PC sin ningún proceso de instalación.

## 3.3 ESTUDIO DE LAS TECNOLOGÍAS

En este apartado se van a analizar las diferentes tecnologías existentes que pueden solucionar el problema planteado en el punto anterior. Se hará un repaso de todas las posibilidades que se tuvieron en cuenta a la hora de realizar el presente Proyecto Fin de Carrera, para finalmente explicar en detalle aquéllas que se eligieron como mejor solución. El estudio de las tecnologías abarca los siguientes aspectos:

- o Plataforma software sobre la que se ejecutará la aplicación
- o Patrones de diseño de la arquitectura de la aplicación
- o Tecnologías empleadas para la creación del documento automático que genera la aplicación.

### 3.3.1 Posibles soluciones

A continuación se muestran algunas de las alternativas tecnológicas existentes para la realización de este proyecto, enumerando sus ventajas e inconvenientes.

## Lenguaje de Programación

Como se puede ver en el apartado **3.2.2 Requisitos no funcionales**, los requisitos no funcionales números tres y cuatro especifican como lenguaje de programación el lenguaje JAVA. Se hará un diagnostico de la versión más apropiada a utilizar y un estudio de la compatibilidad de todas ellas.

## Entorno de desarrollo

Para el entorno de desarrollo se tuvieron en cuenta las siguientes posibilidades:

- **Eclipse**: Plataforma de programación usada para crear entornos integrados de desarrollo (IDE). Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Su característica más relevante es la gran cantidad de *pluggins* y proyectos disponibles que dotan a Eclipse de una gran capacidad de crecimiento.
- **Netbeans**: NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun Microsystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos. La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados *módulos*. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (*manifest file*) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma

NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

- **Otros:** Existen otras soluciones menos completas pero que también podrían servir para el propósito de este proyecto:
  - Emacs
  - DrJava
  - JBuilder
  - JDeveloper
  - JCreator

### **Estándares abiertos de documentación**

Según el requisito funcional número diez, que se puede ver en el apartado **3.2.1 Requisitos funcionales**, el documento generado por la aplicación debe definirse en un estándar libre de documentación.

Según la **Comisión IDABC de la Comunidad Europea [2]**, cuatro son las características mínimas que una especificación y sus documentos de apoyo deben tener para ser denominados estándares abiertos:

1. El estándar ha sido adoptado y es mantenido por una entidad sin ánimo de lucro, y su sucesivo desarrollo tiene lugar sobre la base de un proceso de decisión abierto a todas las partes interesadas (consenso o decisión por mayoría, etc.).
2. El estándar se ha publicado y el documento con la especificación del mismo se encuentra disponible de forma gratuita o bien por un precio simbólico. Se debe permitir a cualquiera su copia, distribución y uso sin cargo o con un precio simbólico.

3. La propiedad intelectual -por ejemplo, posibles patentes presentes- del estándar (o de alguna de sus partes) se ofrece de forma irrevocable libre de regalías (*royalty-free basis*).
4. No hay restricciones en cuanto a la reutilización del estándar.

En la siguiente tabla se muestra una comparativa muy básica entre estándares abiertos y cerrados:

**Tabla 3-1: Estándar abierto vs Estándar cerrado**

ESTÁNDAR CERRADO	ESTÁNDAR ABIERTO
Creado y mantenido por una entidad privada	Adoptado y mantenido por una entidad sin ánimo de lucro
Uso restringido	Disponible de forma gratuita o bien por un precio simbólico
No contrastado	Propiedad intelectual libre de regalías
Prohibida su redistribución	Sin restricciones en cuanto a su reutilización

Las alternativas que se han manejado a la hora de elaborar el Proyecto son las siguientes:

- **OpenDocument (ODF): El Formato de Documento Abierto para Aplicaciones Ofimáticas de OASIS** (en inglés, *OASIS Open Document Format for Office Applications*), también referido como formato **OpenDocument (ODF)**, es un formato de fichero estándar para el almacenamiento de documentos ofimáticos tales como hojas de cálculo, memorandos, gráficas y presentaciones. Aunque las especificaciones fueron inicialmente elaboradas por Sun, el estándar fue desarrollado por el comité técnico para Open Office XML de la organización OASIS y está basado en un esquema XML inicialmente creado e implementado por la suite ofimática OpenOffice.org.

El estándar fue desarrollado públicamente por un grupo de organizaciones, es de acceso libre, y puede ser implementado por cualquiera sin restricción. Con ello, Open Document es el primer estándar para documentos ofimáticos implementado por distintos competidores, visado por organismos de estandarización independiente y susceptible de ser implementado por cualquier proveedor.

- **PDF:** **PDF** (acrónimo del inglés *Portable Document Format*, formato de documento portátil) es un formato de almacenamiento de documentos, desarrollado por la empresa Adobe Systems. Este formato es de tipo compuesto (imagen vectorial, mapa de bits y texto). Está especialmente ideado para documentos susceptibles de ser impresos, ya que especifica toda la información necesaria para la presentación final del documento, determinando todos los detalles de cómo va a quedar, no requiriéndose procesos anteriores de ajuste ni de maquetación. Cada vez se utiliza más también como especificación de visualización, gracias a la gran calidad de las fuentes utilizadas y a las facilidades que ofrece para el manejo del documento, como búsquedas, hiperenlaces, etc.
- **Office Open XML (OOXML):** Es un formato de archivo usado para representar hojas de cálculo, diagramas, presentaciones y documentos de texto. Un archivo Office Open XML contiene principalmente datos basados en XML comprimidos en un contenedor *zip*.

En el 2006, la especificación de Office Open XML se convierte en formato abierto y estándar Ecma International. La especificación fue desarrollada originalmente por Microsoft para reemplazar sus formatos binarios, además de los formatos de archivo basados en XML de Office 2003. Comenzando desde Microsoft Office 2007, los formatos de archivo Office Open XML (ECMA-376) son el formato predeterminado para guardar documentos de Microsoft Office, actualmente la suite ofimática más usada del mercado. Microsoft Office 2010 será la primera versión en implementar el formato de archivo Office Open XML compatible con el nuevo estándar.

### **3.3.2 Soluciones adoptadas**

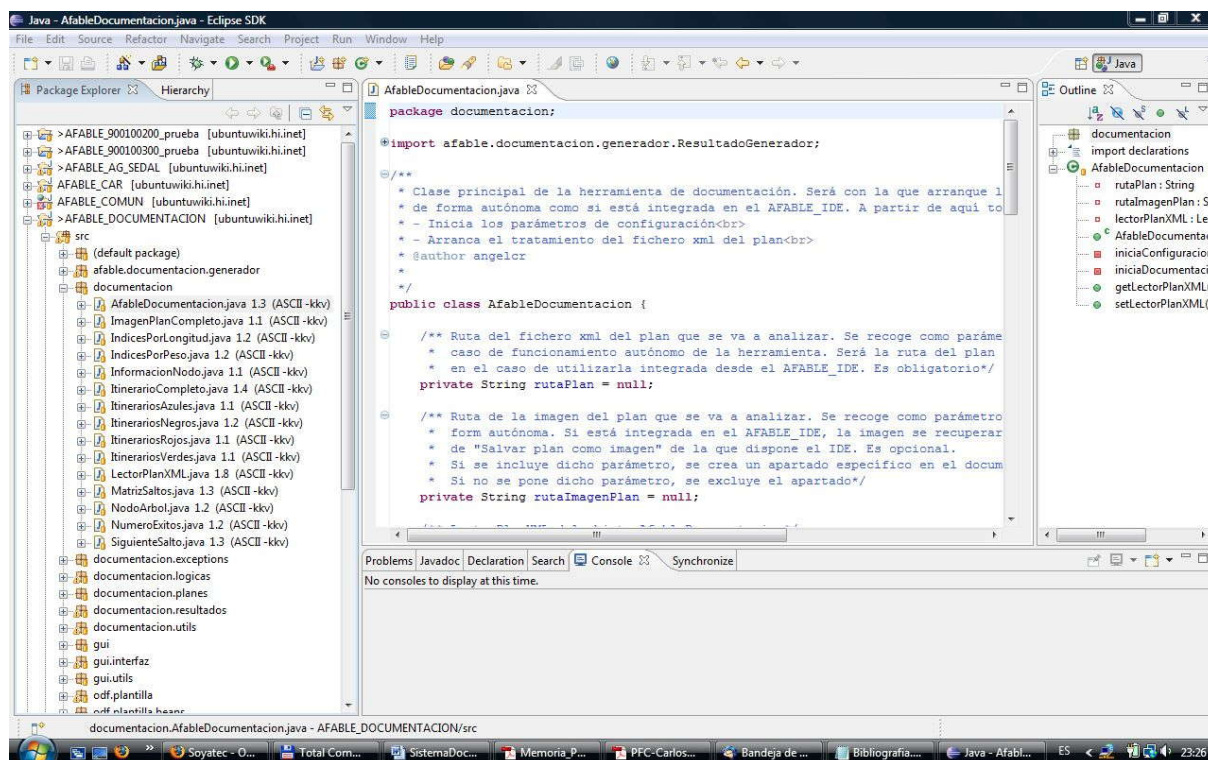
#### **3.3.2.1 Lenguaje de programación JAVA**

Java es el lenguaje de programación elegido para el desarrollo del proyecto según los requisitos no funcionales vistos en apartados anteriores. Se han hecho pruebas para las últimas versiones de la máquina virtual. Se ha probado con éxito para JDK 1.4.2, JDK 1.5 y JDK 1.6. Finalmente, se ha optado por la versión 1.5, puesto que también era requisito no funcional de la aplicación mantener la misma versión de Java para todos los componentes del Sistema AFABLE.

#### **3.3.2.2 Plataforma de desarrollo ECLIPSE**

La plataforma de desarrollo utilizada para implementar la aplicación ha sido Eclipse 3.2. En la siguiente figura se muestra una vista de la ventana principal de Eclipse:

Figura 3-4: Vista genera de Eclipse 3.2



Aunque todos los entornos integrados de desarrollo presentados en el apartado 3.3.1 Posibles soluciones cumplen las necesidades de la aplicación, los dos principales, Eclipse y Netbeans se han estudiado con mayor atención. Finalmente se ha optado por utilizar Eclipse en base a las siguientes razones:

- Mayor fluidez en la interfaz gráfica
- Mejor integración con otros lenguajes de programación
- Mayor facilidad de uso
- Mejor planteamiento a ejecutar/compilar
- Existencia de pluggins especialmente indicados para la elaboración de este proyecto.
- *eUML2: Pluggin* de modelado UML para Eclipse. Presenta una versión de pago (Studio) y una versión libre (Free).



Sirve entre otras cosas, para generar modelos automáticos tales como diagramas de clases, diagramas de secuencia, casos de uso, etc.

**Figura 3-5: Pluggin eUML2 para Eclipse**



### **3.3.2.3 Estándar de documentación abierto OpenDocument (ODF)**

El estándar elegido para la realización del proyecto ha sido OpenDocument (ODF). En concreto, el documento generado se basará en la versión OpenDocument v1.1, que incorpora características adicionales para resolver e incluir algunas funciones de accesibilidad necesarias. Fue aprobado como estándar OASIS el 1 de Febrero de 2007.

#### **Tipos de ficheros**

En la siguiente figura se recogen los tipos de ficheros soportados por el estándar. Para el presente proyecto, se utilizarán únicamente ficheros de texto con extensión *.odt*

**Figura 3-6: Tipos de ficheros soportados****Documentos**

Tipo de formato	Extensión	Tipo de MIME
Texto	.odt	application/vnd.oasis.opendocument.text
Hoja de cálculo	.ods	application/vnd.oasis.opendocument.spreadsheet
Presentación	.odp	application/vnd.oasis.opendocument.presentation
Dibujo	.odg	application/vnd.oasis.opendocument.graphics
Gráfica	.odc	application/vnd.oasis.opendocument.chart
Fórmula matemática	.odf	application/vnd.oasis.opendocument.formula
Base de datos	.odb	application/vnd.oasis.opendocument.database
Imagen	.odi	application/vnd.oasis.opendocument.image
Documento maestro	.odm	application/vnd.oasis.opendocument.text-master

Un fichero OpenDocument es un archivo comprimido en ZIP, que contiene varios ficheros y directorios:

**Figura 3-7: Contenedor ZIP de un fichero OpenDocument**

El formato OpenDocument ofrece una clara separación entre el contenido, la disposición de éste en el documento y los metadatos. Los componentes más notables del formato son los siguientes:

- **content.xml**. Este es el fichero más importante. Almacena el contenido real del documento (excepto los datos binarios como las imágenes). El formato de base utilizado fue inspirado por el HTML, aunque es bastante más complejo que éste, y debería ser razonablemente legible para un humano:

**Figura 3-8: Fichero content.xml**

```
<text:h text:style-name="Heading_2">Título</text:h>
<text:p text:style-name="Text_body" />
<text:p text:style-name="Text_body">
  Éste es un párrafo. La información sobre el formato
  se almacena en el fichero de estilo.
  La marca vacía text:p que se ve más arriba es un
  párrafo en blanco (una línea vacía).
</text:p>
```

- **styles.xml**. OpenDocument hace un uso intensivo de los estilos para el formateo y disposición del contenido. La mayor parte de la información de estilo se almacena en este fichero (aunque hay parte que aparece en el fichero content.xml). Hay diferentes tipos de estilo, que incluyen los siguientes:
  - Estilos de párrafo
  - Estilos de página
  - Estilos de carácter
  - Estilos de marco
  - Estilos de lista

El formato OpenDocument es único en el hecho de que no se puede evitar el uso de estilos para formatear los documentos. Incluso el formateo "manual" se realiza mediante estilos (que la aplicación ofimática debe crear dinámicamente según sean necesarios).

- **meta.xml**. Contiene los metadatos del documento. Por ejemplo, el autor, la identificación de la última persona que lo modificó, la fecha de última modificación, etc. El contenido tiene un aspecto similar a éste:

**Figura 3-9: Fichero meta.xml**

```

<meta:creation-date>2003-09-10T15:31:11</meta:creation-date>
<dc:creator>Daniel Carrera</dc:creator>
<dc:date>2005-06-29T22:02:06</dc:date>
<dc:language>es-ES</dc:language>
<meta:document-statistic
  meta:table-count="6" meta:object-count="0"
  meta:page-count="59" meta:paragraph-count="676"
  meta:image-count="2" meta:word-count="16701"
  meta:character-count="98757" />

```

- **settings.xml**. Este fichero incluye propiedades como el factor de zoom o la posición del cursor que afectan a la apertura inicial del documento, pero no son contenido ni afectan a la disposición de éste en el documento.
- **Pictures/**. Esta carpeta contiene todas las imágenes del documento. El fichero content.xml contiene referencias a ellas mediante el uso de la etiqueta `<draw:image>`, similar a la etiqueta `<img>` de HTML. A continuación se da un ejemplo de una de estas referencias:

**Figura 3-10: Ejemplo de código de una imagen**

```

<draw:image
  xlink:href="Pictures/100000000000005E80000049F21F631AB.tif"
  xlink:type="simple" xlink:show="embed"
  xlink:actuate="onLoad" />

```

La información de posicionamiento (anchura, posición, etc.) se da mediante una etiqueta `<draw:frame>` que contiene a su vez la etiqueta `<draw:image>`.

La mayoría de las imágenes se guardan en su formato original (GIF, JPEG, PNG), aunque los mapas de bits se convierten a PNG por cuestiones de tamaño.

- ***mimetype***. Se trata de un fichero con una única línea que contiene el tipo MIME del documento. Una implicación de esto último es que, en realidad, la extensión del nombre del fichero es indiferente del formato real, toda vez que la que prevalece es la definida por este fichero. Así, la extensión del fichero se utiliza sólo para facilitar la identificación del tipo de fichero por parte del usuario.

### **Aplicaciones que soportan el formato OpenDocument (ODF)**

El formato OpenDocument es utilizado indistintamente en aplicaciones de software libre o software propietario. Esto incluye suites ofimáticas (de tipo tradicional o basadas en Web) y aplicaciones individuales como procesadores de texto, programas de manejo de hojas de cálculo, presentaciones y datos. Algunas de estas aplicaciones incluyen:

- [Abiword](#) 2.4 para lectura y a partir de la 2.4.2 para lectura y escritura
- [Corel WordPerfect Office X4](#)
- [Google Docs](#)
- [IBM Lotus Symphony](#)
- [eZ publish](#) 3.6, con la extensión para OpenOffice
- [Knomos](#) case management 1.0
- [KOffice](#)
- [Microsoft Office 2007 SP2](#)
- [NeoOffice.org](#)
- [ODFReader](#) un plugin para ver los ODF desde Firefox (versiones 1.5 y 2).

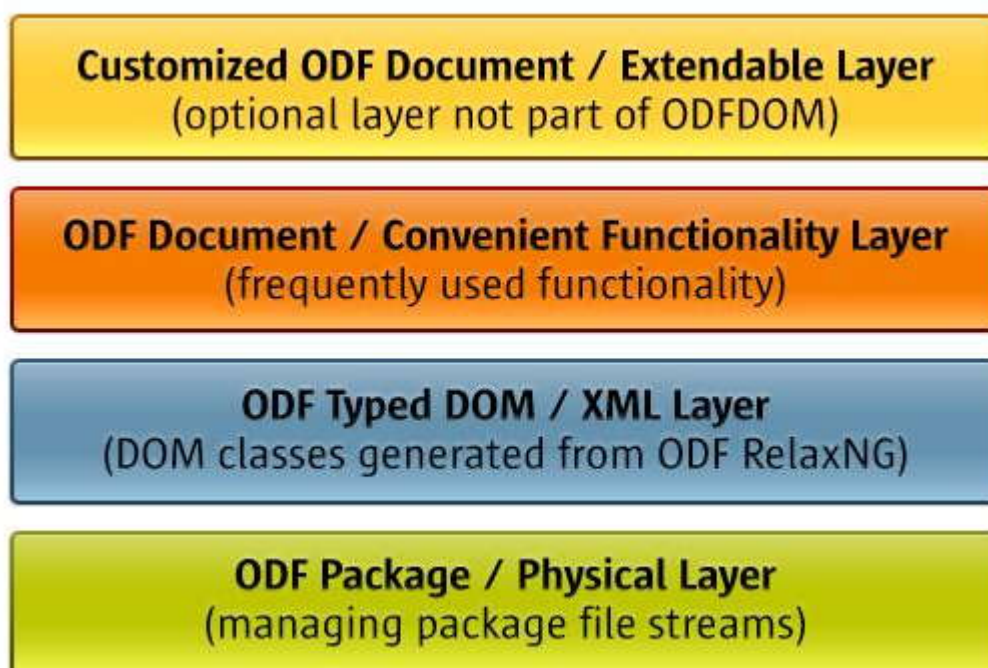
- [OpenOffice.org](http://OpenOffice.org). Suite ofimática gratuita. Será el utilizado para visualizar los documentos en el Proyecto.
- [Scribus](http://Scribus) 1.2.2, pudiendo importar texto y gráficos OpenDocument
- [SoftMaker Office](http://SoftMaker Office)
- [Sun Microsystems StarOffice](http://Sun Microsystems StarOffice)
- [TextMaker](http://TextMaker) 2005 beta
- [Visioo-Writer](http://Visioo-Writer) permite que usuarios sin una suite ofimática adecuada puedan ver los documentos en este formato
- [WordPad](http://WordPad) a partir de la versión incluida con [Windows 7](http://Windows 7) permite leer y escribir textos en formato OpenDocument.
- [Zoho Office Suite](http://Zoho Office Suite)

## **API ODFDOM**

El API ODFDOM implementa un framework para Java, a partir de una serie de paquetes, interfaces y clases que permiten crear, acceder y manipular ficheros odf. Se basa en un modelo de capas, que permite un acceso robusto y modularizado a cada uno de los ficheros basados en OpenDocument.

El modelo de capas se presenta en la siguiente figura:

**Figura 3-11: Modelo de capas del API ODFDOM**



- **ODF Package / Physical Layer (Capa física):** Es la primera capa del modelo. Proporciona acceso directo a los recursos almacenados en el paquete ODF, como son los flujos XML de datos que componen los ficheros, las imágenes u otro tipo de objetos embebidos.

En este nivel, un documento se representa por un conjunto de recursos comprimidos en un paquete. Por ejemplo, un documento de texto ODF llamado “*MyVacation.odt*” debe contener los ficheros que se muestran en la siguiente figura:



**Figura 3-12: Ficheros de la capa física**



Los principales requisitos de esta capa son:

- Comprimir y descomprimir los flujos de ficheros dentro del paquete
- Enlazar todos los flujos de ficheros en el archivo /META-INF/manifest.xml
- Incluir en el paquete un fichero descomprimido de tipo 'mimetype' con el tipo MIME del documento
- **ODF Typed DOM / XML Layer (Capa XML):** Es la segunda capa del modelo. Representa los elementos ODF XML de los flujos XML de los diferentes ficheros como pueden ser el content.xml o el styles.xml. Siguiendo los conceptos DOM, la capa provee una clase por cada elemento ODF XML definido por la especificación ODF y su gramática (RelaxNG schema).

En este nivel, todos los ficheros xml siguen el modelo definido en el API de DOM (Document Object Model) aunque, sólo los ficheros content.xml y styles.xml tienen implementadas cada una de sus entradas XML como clases ODF. La implementación para el resto de ficheros queda relegada al siguiente nivel.

A continuación se muestra cómo se representa en clases ODF, el código XML correspondiente a una tabla dentro del documento:



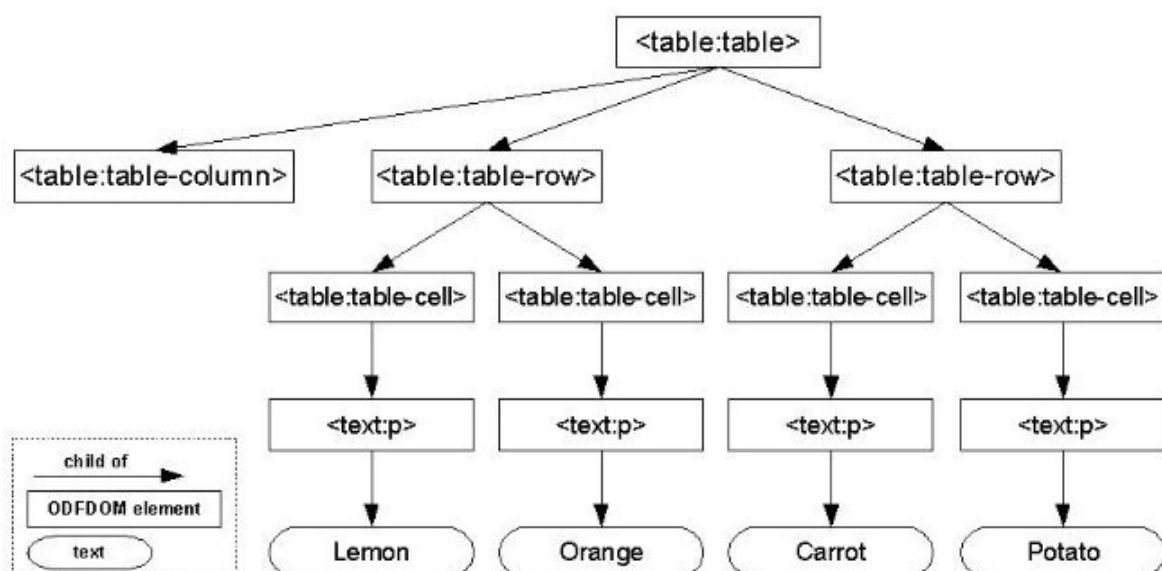
**Figura 3-13: Ejemplo código XML de una tabla en el documento**

```

<table:table table:name="Table - fruits vs. vegies">.
  <table:table-column table:number-columns-repeated="2"/>.
  <table:table-row>.
    <table:table-cell>.
      <text:p>Lemon</text:p>.
    </table:table-cell>.
    <table:table-cell>.
      <text:p>Orange</text:p>.
    </table:table-cell>.
  </table:table-row>.
  <table:table-row>.
    <table:table-cell>.
      <text:p>Carrot</text:p>.
    </table:table-cell>.
    <table:table-cell>.
      <text:p>Potato</text:p>.
    </table:table-cell>.
  </table:table-row>.
</table:table>.

```

Este código XML será mapeado a una estructura de clases como la siguiente:

**Figura 3-14: Ejemplo estructura de clases de una tabla en el documento**

- **ODF Document / Convenient Functionality Layer (Capa de funciones):** El objetivo de esta capa es dotar a los usuarios del API de una funcionalidad de más alto nivel. Para conseguir esto, es necesario una mayor representación de clases y objetos. La representación mediante clases de los elementos XML permite, en este nivel, un control más extenso de los objetos ODF y sus atributos. Normalmente, las clases del nivel anterior extenderán de las clases de este nivel, por lo que se recogerá parte de la misma lógica.
- **Customized ODF Document / Extendable Layer (Capa extendida):** Es la capa de mayor nivel en la jerarquía. Todavía no está implementada en el proyecto ODFDOM. Tiene como objetivo, en un futuro, reemplazar, modificar y actualizar elementos ODF. Por ejemplo, cambiar el estilo o tamaño por defecto de una tabla.

A continuación, se recoge en una tabla, a modo de resumen, los motivos fundamentales por los que se ha elegido el estándar abierto OpenDocument (ODF):

**Tabla 3-2: Ventajas del OpenDocument (ODF)**

VENTAJAS DEL OPENDOCUMENT (ODF)
Gran capacidad de integración con Java al basarse en XML
Amplia documentación del estándar en su versión 1.1
Suite ofimática libre y extendida como es OpenOffice
<i>Plugins</i> para visualización de documentos ODF para Microsoft Office y Mozilla Firefox
Existencia del API ODFDOM para el manejo de ficheros ODF

---

## 4 ANÁLISIS DEL SISTEMA

### 4.1 INTRODUCCIÓN

En esta sección se va a realizar un análisis formal de la aplicación que compone el Proyecto Fin de Carrera. En primer lugar, se explicará los conceptos generales asociados a Red Inteligente necesarios para entender la aplicación. Se hará un repaso de los diferentes Módulos de Lógica que pueden componer un Plan de Abonado, indicando que información relativa a cada uno de ellos deberá extraer la aplicación para incorporarla al documento automático del Plan. Una vez analizados los diferentes nodos y los tipos de transiciones que pueden existir entre ellos se hará un estudio de los diferentes Casos de Uso de la herramienta. Para ello, se propondrán diagramas UML que ayuden a entender la interacción del usuario con la aplicación para cada una de las acciones que se pueden llevar a cabo.

### 4.2 PLAN DE ABONADO VOICEXML

Se denomina **Plan de Abonado VoiceXML** a la descripción formal del comportamiento de un servicio 90X creado mediante el sistema AFABLE.

Una aplicación VoiceXML consiste en una secuencia de diálogos de voz interactivos con el usuario final contruidos sobre las tecnologías de conversión texto/voz, voz pregrabada y reconocimiento de voz y tonos DTMF. Estas tecnologías se combinan con elementos propios de lógica de programación, elementos de telefonía y elementos especiales de acceso a recursos externos, tales como bases de datos o aplicaciones externas como por ejemplo Web Services, con el fin de proporcionar los servicios deseados al usuario final.

La representación formal de un Plan de Abonado VoiceXML consiste en un fichero XML válido de acuerdo a:

- La especificación sintáctica definida por el “*XML Schema PlanAbonadoVoiceXML V1.0*” recogido en un fichero *.xsd*.
- Las restricciones semánticas adicionales impuestas por la herramienta AFABLE IDE de creación de planes de abonado para que éstos sean funcionalmente correctos.

A continuación se definen algunos conceptos directamente relacionados con el diseño gráfico de los Planes de Abonado VoiceXML.

#### 4.2.1 Grafo del Plan de Abonado VoiceXML

Se denomina “**Grafo**” a la representación gráfica de un Plan de Abonado VoiceXML mediante un árbol de decisión que contiene la información esencial del mismo, fácil de entender por personal no especializado en servicios IVR, como por ejemplo, el titular del servicio 90X.

El Grafo de un Plan de Abonado VoiceXML, en el sistema AFABLE IDE, se dibuja mediante un diagrama de flujo formado por un conjunto de nodos denominados “**Módulos de Lógica**” que se relacionan entre sí a través de “**transiciones**” con el fin de representar de una forma sencilla la funcionalidad del servicio 90X deseado.

#### 4.2.2 Módulos de Lógica (ML)

Se denomina “**Módulo de Lógica**” a cada uno de los nodos que conforman un Plan de Abonado y que determina un comportamiento funcional concreto dentro del servicio vocal que se ejecuta en los elementos especiales de la Red Inteligente. Tienen una representación gráfica concreta que los identifica en el grafo del plan construido con la herramienta AFABLE IDE.

#### 4.2.3 Transiciones entre Módulos de Lógica

Las transiciones entre módulos de lógica determinan el orden de ejecución de cada uno de los módulos de lógica que componen un plan de abonado.

Un módulo de lógica (dependiendo de la funcionalidad concreta que implementa) puede tener definidas varias transiciones diferentes a otros módulos de lógica del grafo del plan, en función de:

- La finalización correcta de la ejecución de la lógica
- La finalización con error de la ejecución de la lógica
- La ocurrencia de un evento determinado. En el caso concreto de los módulos de lógica que implementan interacciones con el usuario se pueden definir transiciones diferentes en función de ciertos eventos producidos durante la interacción, como es, por ejemplo, el superar el máximo número de intentos para obtener un dato de entrada correcto mediante reconocimiento DTMF o voz de acuerdo a las gramáticas activas.

### 4.3 DESCRIPCIÓN DE LOS MÓDULOS DE LÓGICA

En el apartado anterior se han explicado los conceptos básicos relativos a la definición de Plan de Abonado, incluyendo la caracterización de los Módulos de Lógica (ML) y las transiciones que se producen entre ellos. El módulo AFABLE DOCUMENTACIÓN, objetivo de este Proyecto, deberá tener en cuenta cada uno de estos nodos, con el fin de mostrar en los documentos automáticos que se generen la información más relevante de su funcionalidad.

Los nodos de un Plan de Abonado se agrupan dentro de lógicas específicas. Cada módulo de lógica determina un comportamiento funcional concreto dentro del servicio vocal y se clasifican de la siguiente forma en función del comportamiento funcional que representan:

- **Lógicas Básicas de Interacción con el Usuario.** Representan elementos que implementan la funcionalidad de interacción con el usuario: Emisión de locuciones, entrada de datos por reconocimiento de voz y tonos DTMF de acuerdo a gramáticas predefinidas, selección de una opción de menú y grabación de mensajes de voz.

- *Nodo Locución*. Se encarga de emitir una locución al usuario.
- *Nodo Menú*. Permite elegir al usuario una opción de las definidas en el menú.
- *Nodo Dato de Entrada*. Obtiene un valor de entrada dado por el usuario del sistema.
- *Nodo Grabar Mensaje voz*. Permite grabar un mensaje dicho por el usuario

**Figura 4-1: Nodos de Interacción con el Usuario**



- **Lógicas Básicas de Telefonía.** Representan elementos que implementan la funcionalidad de telefonía: Transferencia de llamadas
  - *Nodo Transferir Llamada*. Transfiere una llamada a otro número de teléfono.

**Figura 4-2: Nodos de Telefonía**



- **Lógicas Básicas de Programación.** Representan elementos que tienen una funcionalidad típica de los lenguajes de programación: Asignación de variables, ejecución condicional if o switch e inicio y fin de la ejecución del servicio vocal.

- *Nodo Asignar Variable*. Permite dar un valor a una variable previamente definida en el Plan de Abonado
- *Nodo if*. Permite ejecutar una rama u otra del plan de abonado dependiendo de una condición que se evaluará como verdadera o falsa.
- *Nodo switch*. Permite ejecutar una rama u otra del plan de abonado dependiendo de una condición que se evaluará como un valor que determina la rama a ejecutar.
- *Nodo rutina ECMAScript*. Permite ejecutar algoritmos programados en lenguaje ECMAScript.
- *Nodo Inicio*. Es el primero del Plan de Abonado. Debe existir siempre.
- *Nodo Fin*. Es el último nodo del Plan de Abonado.

**Figura 4-3: Nodos básicos de Programación**



- **Lógicas Especiales.** Representan elementos de interacción con otros sistemas externos a AFABLE para obtener un resultado a utilizar en la lógica del servicio VoiceXML. Son el envío de un SMS o MMS a un móvil, envío de un correo electrónico, envío de un mensaje de voz al CAR, consulta y modificación de una base de datos, invocación de un servicio Web externo, geolocalizar un teléfono o crear una gramática online a partir de la información obtenida de una base de datos.

- *Nodo Enviar Correo*. Envía un mensaje de correo electrónico.
- *Nodo Enviar SMS*. Envía un mensaje de texto a un móvil.
- *Nodo Enviar MMS*. Envía un mensaje multimedia a un móvil.
- *Nodo Envío mensaje al CAR*. Envía un mensaje de voz a un contestador.
- *Nodo Geolocalizar Teléfono*. Permite localizar geográficamente un número de teléfono, generalmente el del llamante.
- *Nodo Consultar Base de Datos*. Permite obtener información de una Base de Datos local o remota a través de una interfaz JDBC.
- *Nodo Modificar Base de Datos*. Permite añadir o modificar información de una base de datos local o remota a través de una interfaz JDBC.
- *Nodo Invocar Servicio Externo*. Permite realizar una petición a un servicio externo a través de una interfaz XML sobre HTTP y obtener la respuesta a la misma.
- *Nodo Gramática*. Permite utilizar gramáticas previamente definidas en el Plan de Abonado.

**Figura 4-4: Nodos basados en Lógicas Especiales**



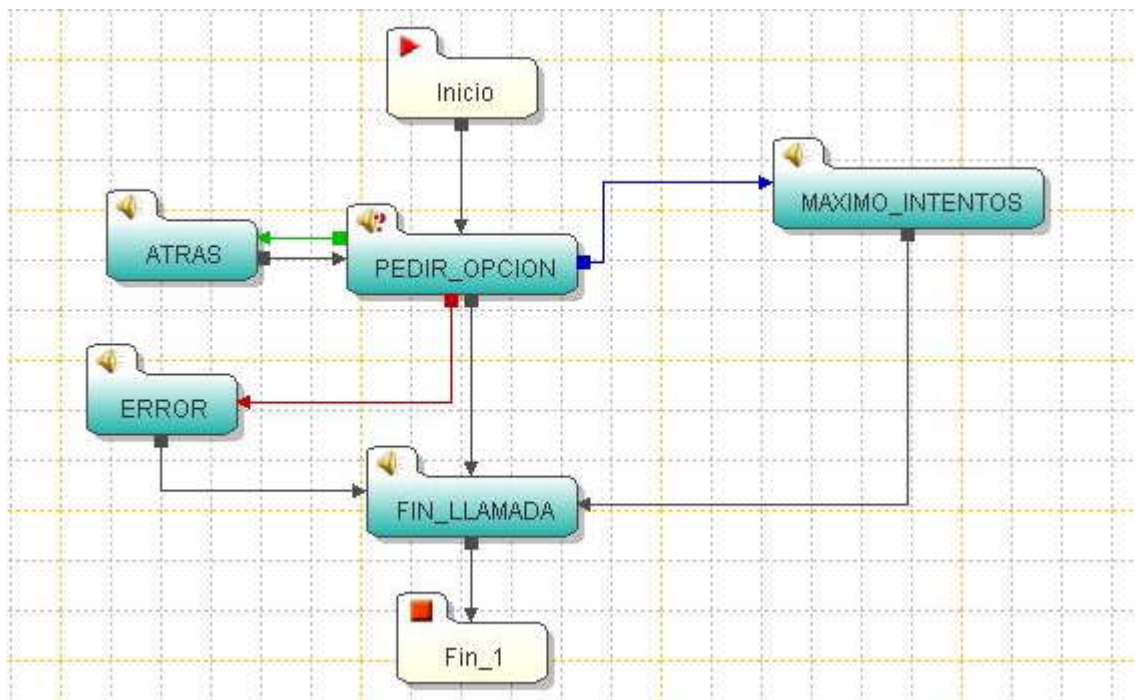


El segundo elemento que compone el grafo del Plan de Abonado son las **Transiciones**. Todos los nodos tienen unos campos que definen transiciones. Cualquiera de estos campos presenta una lista desplegable con los nodos disponibles a los que se puede realizar la transición, incluido el propio nodo. Los atributos que definen las transiciones dependen del tipo de nodo origen. Por ejemplo, para un Nodo de tipo Dato de Entrada, las transiciones que se pueden definir son:

- Atrás
- Máximo de intentos
- Error
- Dato obtenido

Para definir cada una de estas transiciones se debe elegir de la lista de Nodos del Grafo cual es el Nodo destino. Dependiendo del tipo de transición las líneas que unen los distintos nodos tendrán un color diferente:

- Las líneas correspondientes a **transiciones provocadas por el fin normal** de ejecución del nodo se dibujarán en color **negro**.
- Las líneas correspondientes a transiciones provocadas por agotar el **número máximo de intentos** para obtener un dato correcto desde el usuario se dibujarán en color **azul**.
- Las líneas correspondientes a transiciones derivadas de la ocurrencia de **errores** que impiden la ejecución correcta del módulo de lógica se dibujarán en color **rojo**.
- Las líneas correspondientes a transiciones derivadas de la ocurrencia del **evento atrás** se dibujarán en color **verde**.

**Figura 4-5: Ejemplo de transiciones en un Plan de Abonado**

#### 4.3.1 NODO INICIO

Es el primer Módulo de Lógica que aparece ya predefinido en cualquier plan y su misión es identificar el primer Módulo de Lógica que se debe ejecutar en el plan. Su nombre no puede ser modificado y siempre se llamará “Inicio”.

**Icono del nodo:**

#### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)

- *Transición. (ÉXITO).*

#### 4.3.2 NODO FIN

Este Módulo de Lógica representa la finalización de la ejecución del Plan de Abonado. Es el último nodo que se debe ejecutar en cualquier plan y debe existir obligatoriamente como último nodo a ejecutar en todas las ramas para que sea semánticamente correcto.

**Icono del nodo:** 

##### **Información recogida en el Documento Automático**

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo (opcional)*

#### 4.3.3 NODO DE CONDICIÓN IF

Determina la ejecución de una u otra rama funcional en el diagrama de flujo del plan de abonado en función del cumplimiento o no de la condición predeterminada

**Icono del nodo:** 

##### **Información recogida en el Documento Automático**

- *Nombre de nodo*
- *Tipo de nodo*

- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO).
- *Condición*
- *Opción de salida* (CIERTO, FALSO)

#### 4.3.4 NODO SWITCH

Determina la ejecución de una u otra rama en el grafo de decisión del plan de abonado en función de un valor obtenido tras la ejecución de una expresión.

Icono del nodo: 

##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO).
- *Expresión*
- *Opción elegida en el SWITCH*

#### 4.3.5 NODO RUTINA ECMAScript

La finalidad de este elemento es permitir la incorporación de algoritmos de programación (editados directamente en lenguaje ECMAScript) como parte de la lógica de un Plan de Abonado VoiceXML. Se utilizará en los casos en que dicha lógica sea difícil o inviable de definir mediante otros módulos de lógicas.

**Icono del nodo:** 

#### **Información recogida en el Documento Automático**

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición.* (ÉXITO).
- *Script*

#### **4.3.6 NODO ASIGNAR VARIABLE**

La finalidad de este elemento es poder crear una variable y asignarle un determinado valor para su posterior uso en el Plan de Abonado.

**Icono del nodo:** 

#### **Información recogida en el Documento Automático**

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición.* (ÉXITO).
- *Nombre de la variable*
- *Valor de la variable*

#### 4.3.7 NODO LOCUCIÓN

Este módulo de lógica permite añadir al buffer de reproducción de voz, voz sintética (obtenida mediante un proceso de conversión texto/voz) y locuciones grabadas, que se emitirán en el momento que se ejecute el nodo. También, se emitirán locuciones en los nodos de interacción con el usuario, que requieran una entrada de datos por parte de éste (Nodo Menú, Nodo Dato de Entrada o Nodo de Grabación de Voz).

Icono del nodo: 

##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO, MAX\_INTENTOS, ERROR, ATRÁS).
- *Información sobre si la locución puede ser interrumpida por el usuario o no.*
- *Número de elementos de la locución.*
- *Descripción de cada uno de los elementos de audio que componen la locución.*

#### 4.3.8 NODO MENÚ

Este módulo de lógica permite obtener mediante el reconocimiento de tonos **DTMF** o **voz**, la elección de una opción de un menú. Según la opción elegida se producirán transiciones a diversos nodos.

Icono del nodo: 

#### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO, MAX\_INTENTOS, ERROR, ATRÁS).
- *Tipo de reconocimiento*. Indica el tipo de reconocimiento que se admite para obtener la opción desde el usuario: *dtmf*, *voz* o *ambos*.
- *Opción elegida del menú*.

#### 4.3.9 NODO DATO ENTRADA

Este Módulo de Lógica permite obtener un dato de entrada del usuario y guardarlo en una variable cuyo nombre coincide con el nombre del nodo. El dato se obtiene por reconocimiento de voz o tonos DTMF de acuerdo a las gramáticas configuradas en dicho nodo. Opcionalmente, este nodo permite incorporar la funcionalidad necesaria para **validar** el dato mediante un algoritmo y para **confirmar** con el usuario que el dato introducido es correcto.

Icono del nodo: 

#### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)

- *Transición*. (ÉXITO, MAX\_INTENTOS, ERROR, ATRÁS).
- *Algoritmo de validación* (opcional)
- *Nivel de certidumbre* (1 a 10). Valor entre 1 y 10 que define el nivel de exactitud requerido en el reconocimiento de las opciones. A valores más altos la probabilidad de reconocer una opción correcta será menor.

#### 4.3.10 NODO DE GRABACIÓN DE MENSAJES

Este módulo de lógica permite grabar un mensaje de voz dicho por el usuario, de contenido libre.

Icono del nodo:



##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO, MAX\_INTENTOS, ERROR).
- *Tiempo máximo de grabación*.
- *Terminar DTMF*. Indica que la grabación se finalizará si el usuario pulsa una tecla produciendo un tono dtmf desde su terminal telefónico.

#### 4.3.11 NODO TRANSFERIR LLAMADA

Este módulo de lógica permite transferir una llamada a otro número de teléfono de la red telefónica.



**Icono del nodo:** 

#### **Información recogida en el Documento Automático**

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición.* (ÉXITO).
- *Teléfono.* Teléfono al que se desea transferir la llamada.

#### **4.3.12 NODO ENVIAR CORREO**

Este módulo de lógica permite enviar un correo electrónico

**Icono del nodo:** 

#### **Información recogida en el Documento Automático**

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición.* (ÉXITO, ERROR).
- *Remitente.* Identificado por su cuenta de correo.
- *Destinatarios.* Identificados por su cuenta de correo.

#### 4.3.13 NODO ENVIAR MENSAJE AL CAR

Este módulo de lógica permite enviar un mensaje al CAR (Contestador Automático en Red) obtenido mediante conversión texto/voz o mediante una grabación de voz.

Icono del nodo: 

##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO, ERROR).
- *Número origen*. Identifica el número origen de la llamada al 90X.
- *Número destino*. Identifica el número destino a cuyo buzón hay que mandar el mensaje de voz.
- *Audio grabado*. Indica cuál es el origen del audio a enviar al CAR.

#### 4.3.14 NODO ENVIAR SMS

Este módulo de lógica permite enviar un mensaje de texto a un teléfono móvil.

Icono del nodo: 

##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*

- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO, ERROR).
- *Número origen*. Identifica el número origen de la llamada al 90X.
- *Número destino*. Identifica el número destino al que queremos mandar el mensaje de texto.
- *Texto del mensaje*.

#### 4.3.15 NODO ENVIAR MMS

Permite enviar un mensaje multimedia a un teléfono móvil.

Icono del nodo: 

##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO, ERROR).
- *Número origen*. Identifica el número origen de la llamada al 90X.
- *Número destino*. Identifica el número destino al que queremos mandar el mensaje multimedia.
- *Audio grabado* (opcional). Indica el valor del audio grabado que acompaña al mms.

#### 4.3.16 NODO GEOLOCALIZAR TELÉFONO

Permite la geolocalización de un teléfono fijo mediante la invocación de una consulta externa.

Icono del nodo: 

##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO, ERROR).
- *Teléfono*. Identifica el número que queremos geolocalizar.

#### 4.3.17 NODO CONSULTAR BASE DE DATOS

Consulta de información sobre una base de datos local (tabla SQL). Es requisito previo a la utilización de este nodo haber definido la tabla de datos.

Icono del nodo: 

##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición*. (ÉXITO, ERROR).

- *Nombre de la tabla.*
- *Cláusula WHERE de la consulta.*

#### 4.3.18 NODO MODIFICAR BASE DE DATOS

Permite insertar o modificar un registro en una Base de Datos, dependiendo de si el registro existe o no. Es requisito previo a la utilización de este nodo haber definido la tabla de datos.

Icono del nodo: 

##### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición.* (ÉXITO, ERROR).
- *Nombre de la tabla.*
- *Cláusulas SET con parejas campo/valor.*

#### 4.3.19 NODO CREAR GRAMÁTICA

Este nodo permite la creación de una gramática en tiempo de ejecución del servicio VoiceXML, con la información obtenida mediante una consulta a una tabla SQL. La consulta SQL deberá arrojar como resultado otra tabla (conjunto de registros) con la información de los vocablos resultantes del reconocimiento y sus respectivas alternativas.

Icono del nodo: 

### Información recogida en el Documento Automático

- *Nombre de nodo*
- *Tipo de nodo*
- *Descripción de nodo* (opcional)
- *Transición.* (ÉXITO, ERROR).
- *Nombre de la gramática.*
- *Modo de entrada de la gramática.* Determina si la gramática es de tipo DTMF o voz.

## 4.4 CASOS DE USO

En ingeniería del software, un **caso de uso** es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.

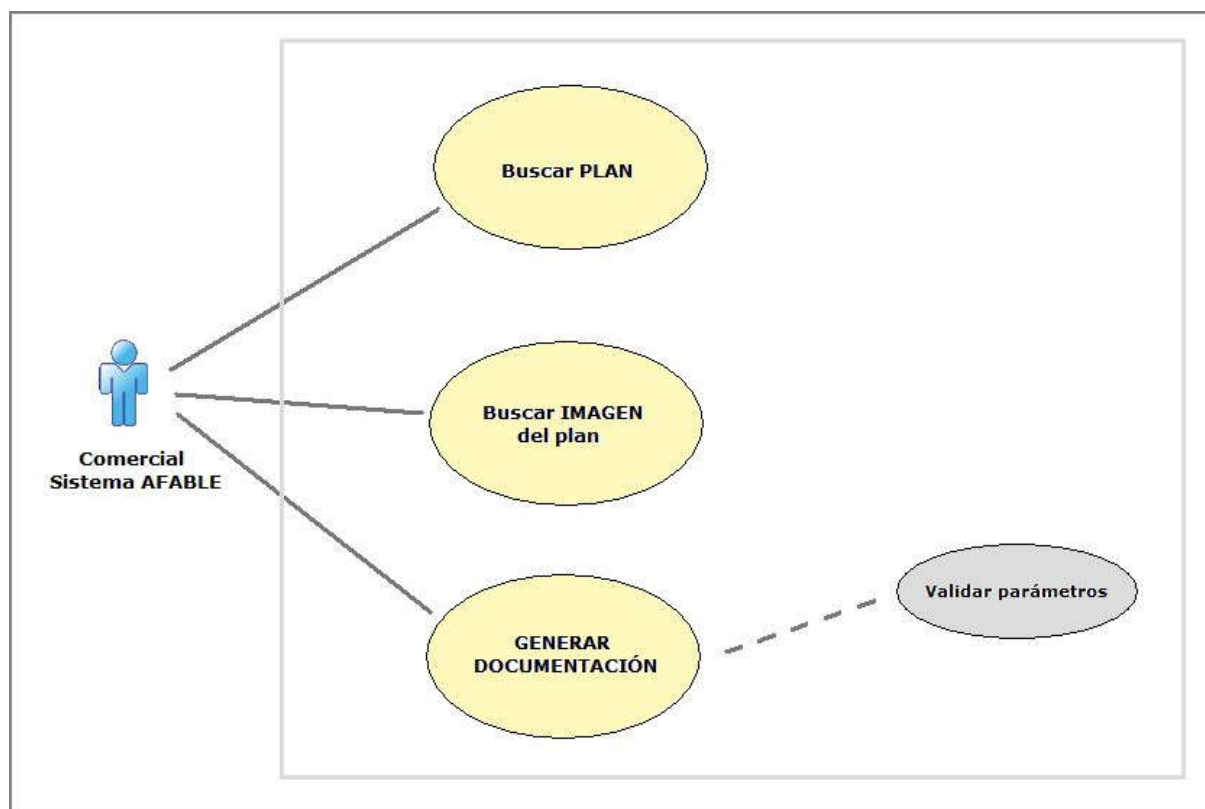
En otras palabras, un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.

### 4.4.1 Casos de uso de primer nivel

En primer lugar, se mostrará el diagrama de casos de uso de primer nivel, que es aquél en el que el actor principal interactúa con el sistema para generar la documentación del Plan de Abonado.

## CASO DE USO: Generación de documentación con la foto del plan opcional.

Figura 4-6: Caso de uso de primer nivel



En el caso de uso de primer nivel, el actor principal selecciona la ruta de un Plan de Abonado, fichero de extensión *.axml*. Opcionalmente, puede introducir también la ruta del grafo del Plan. Posteriormente genera la documentación automática asociada a dicho Plan. Si se introduce la imagen del Plan, el documento automático que se crea contendrá una sección específica dedicada a ella.

### 4.4.1.1 Actores de primer nivel

En el caso de uso desarrollado anteriormente existe un único actor principal. Dicho actor principal será interpretado típicamente por un

**comercial del Sistema AFABLE**, cuyo perfil será el de un usuario experimentado en el manejo de la herramienta gráfica del AFABLE IDE y con amplios conocimientos en la herramienta de documentación AFABLE DOCUMENTACIÓN, objetivo del presente Proyecto Fin de Carrera.

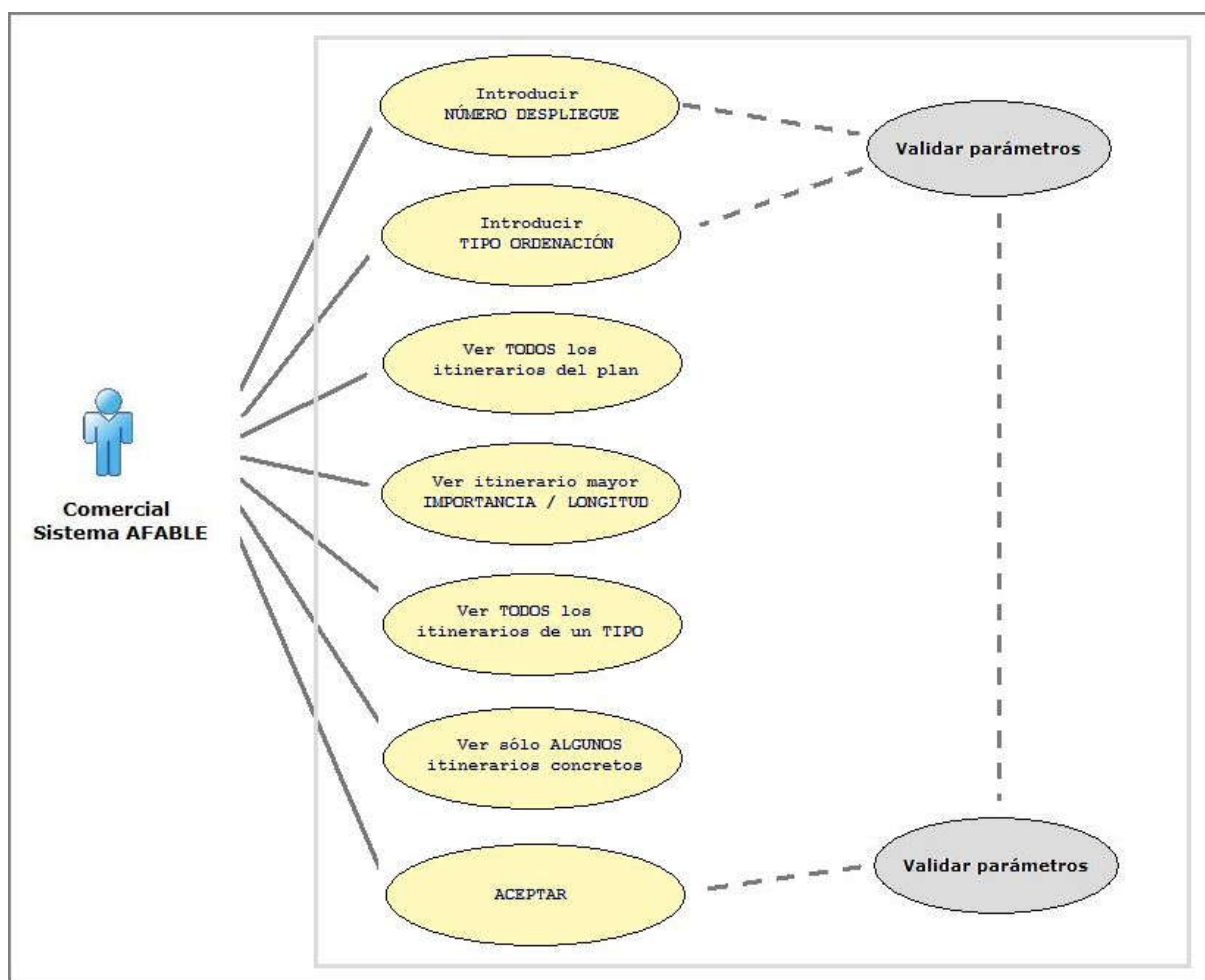
#### **4.4.2 Casos de uso de segundo nivel**

En segundo lugar, se mostrará el diagrama de casos de uso de segundo nivel, que es aquél en el que el actor principal interactúa con el sistema para conseguir una documentación personalizada del Plan de Abonado. Se verán en este punto, todas las posibles opciones de documento automático que se puede generar con la aplicación.



## CASO DE USO: Generar documento automático personalizado

Figura 4-7: Caso de uso de segundo nivel (GENERAR DOCUMENTACIÓN)



En el caso de uso de segundo nivel, vemos como el actor principal debe introducir dos parámetros obligatorios, como son el **número de despliegue** del plan y el **tipo de ordenación**. Se hará una validación de ambos. Se comprobará que el número de despliegue está compuesto de nueve dígitos y que el tipo de ordenación es una de las dos posibles que existen, ordenación por *importancia* u ordenación por *longitud*. Posteriormente, el actor principal podrá personalizar el documento automático generado en base a la elección de las siguientes opciones:

- **Ver TODOS los itinerarios del plan:** Permite crear un documento con todos los itinerarios del Plan de Abonado.
- **Ver itinerario mayor IMPORTANCIA / LONGITUD:** Introduce en el documento automático una sección destinada al itinerario principal del Plan de Abonado.
- **Ver TODOS los itinerarios de un TIPO:** Establece en el documento automático las secciones destinadas a cada uno de los cuatro grupos en que se pueden clasificar los distintos itinerarios del Plan de Abonado
- **Ver sólo ALGUNOS itinerarios concretos:** Permite incluir en la documentación la información relativa a itinerarios concretos del Plan de Abonado.

Una vez establecidas las distintas opciones, el actor principal debe pulsar **ACEPTAR** para terminar de generar el documento automático del Plan de Abonado.

#### **4.4.2.1 Actores de segundo nivel**

En este segundo nivel, se va a distinguir un único actor de uso, que coincide con el de primer nivel:

##### **Comercial del Sistema AFABLE**

Es el actor principal del caso de uso. Sería el mismo que en los casos vistos en el apartado anterior. Se trata de un perfil experimentado, tanto en la herramienta de documentación **AFABLE DOCUMENTACIÓN** como en todo el entorno gráfico constituido por el AFABLE IDE. El comercial del Sistema AFABLE genera un documento automático a medida del titular del Servicio 90X. Así pues, son los deseos del cliente del servicio los que determinan el tipo de documento a generar.

## 4.5 DIAGRAMAS DE SECUENCIA

El **diagrama de secuencia** es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. Un **diagrama de secuencia** muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo. Mientras que el diagrama de casos de uso permite el modelado de una vista del escenario, el **diagrama de secuencia** contiene detalles de implementación del mismo, incluyendo los objetos y clases que se usan para implementar el modelo, y mensajes intercambiados entre los objetos. Típicamente se examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si se tiene modelada la descripción de cada caso de uso como una secuencia de varios pasos, entonces es posible "*caminar sobre*" ellos para descubrir qué objetos son necesarios para que se puedan implementar. Un **diagrama de secuencia** muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas horizontales.

### 4.5.1 Diagrama de secuencia de primer nivel

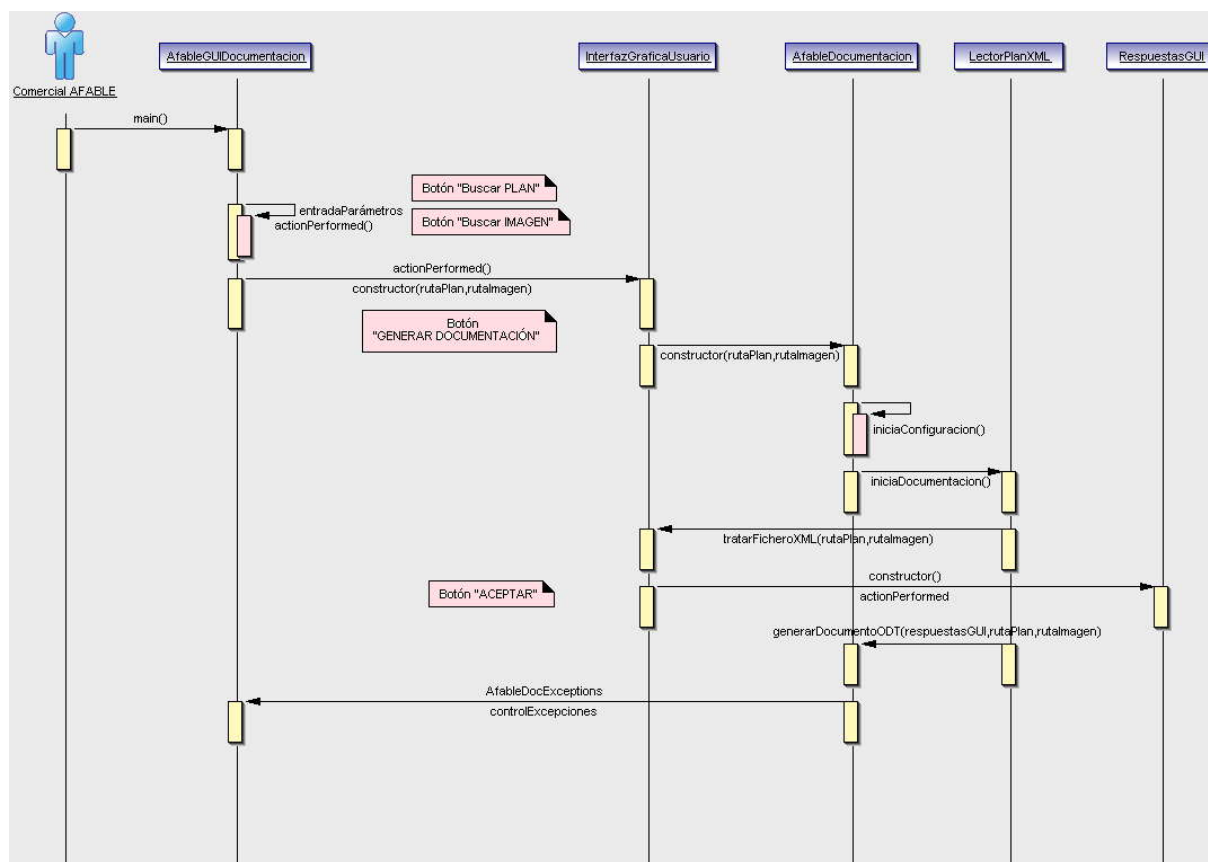
En el diagrama de secuencia de primer nivel se van a desarrollar los dos casos de uso de primer nivel contemplados en el apartado anterior. Como quiera que lo único que diferencia ambos casos es la recogida o no de la imagen del plan como parámetro de entrada, podemos utilizar un único diagrama de secuencia para contemplar los dos.

El diagrama de secuencia se resume en los siguientes pasos:

1. El comercial del sistema AFABLE (actor principal), busca un fichero con extensión *.axml* y una foto del grafo del Plan de Abonado.
2. La clase **AfableGUIDocumentación** presenta la pantalla con los botones de búsqueda "*Buscar PLAN*" y "*Buscar IMAGEN*", y responde al evento que se produce al pulsar sobre el botón "*GENERAR DOCUMENTACIÓN*",

presentando un objeto de tipo **InterfazGraficaUsuario**, que recibe como parámetros la ruta y la imagen del Plan de Abonado.

3. El objeto **InterfazGraficaUsuario** crea en su constructor un objeto de tipo **AfableDocumentación** de modo, que antes de presentar por pantalla el interfaz gráfico ejecuta los siguientes métodos:
  - Método *iniciaConfiguracion()*: Permite la inicialización de constantes y mensajes contenidos en los ficheros de recursos de la aplicación. Además inicializa el sistema de trazas ideado para la herramienta.
  - Método *iniciaDocumentación()*: Realiza la primera parte de la labor de documentación. Crea un objeto de tipo **LectorPlanXML**, a partir del cuál se lee el fichero *.axml* del Plan de Abonado. Se procede a aplicar los algoritmos de ordenación y clasificación de nodos, así como las técnicas “*antibucles*”. Una vez finalizado el proceso, se presenta en pantalla el interfaz gráfico de usuario al que se le pasan como parámetros el número de itinerarios que existe de cada tipo.
4. El objeto **AfableDocumentación**, por medio de su instancia de **LectorPlanXML**, presenta un método que genera el documento personalizado en formato *.odt*. Dicho método es el siguiente:
  - Método *generarDocumentoOdt(respuestasGUIUsuario, rutaPlan, rutaImagen)*: Crea el documento automático personalizado del Plan de Abonado, a partir del objeto **RespuestasGUIUsuario** que se ha recogido una vez que se ha pulsado el botón *ACEPTAR* sobre la clase **InterfazGraficaUsuario**.

**Figura 4-8: Diagrama de secuencia de primer nivel**

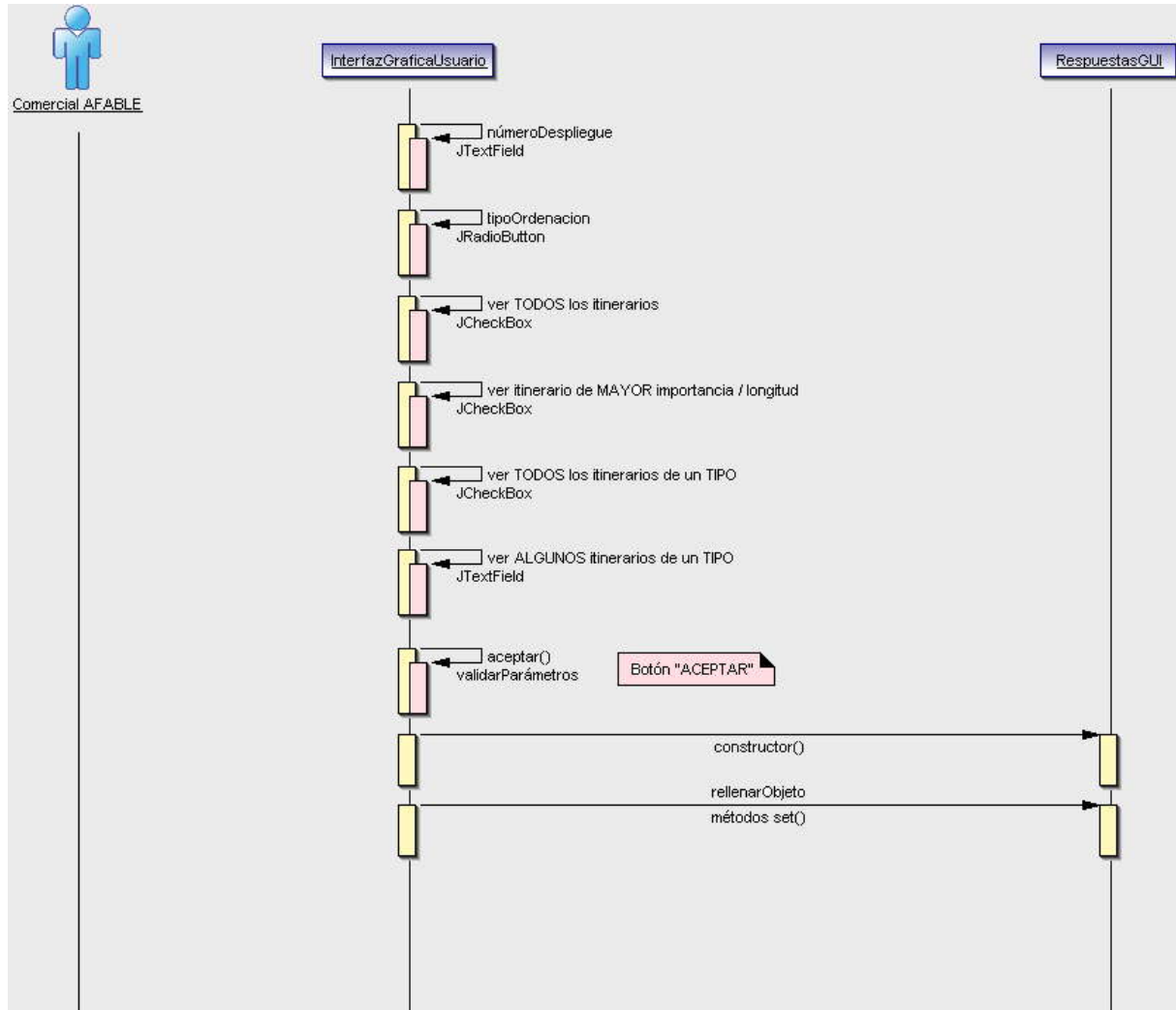
#### 4.5.2 Diagrama de secuencia de segundo nivel

Como hemos visto en el apartado 4.4.2 **Casos de uso de segundo nivel**, los casos de uso de segundo nivel tienen que ver con la capacidad que tiene el titular del Servicio 90X de decidir como quiere que se genere la documentación personalizada de su plan. En este sentido, el diagrama de secuencia que se produce en la aplicación tiene que ver con la forma en la que se genera y se rellena el objeto **RespuestasGUIUsuario**.

Su marco temporal se encuadra en el intervalo que transcurre entre que el objeto **InterfazGraficoUsuario** recibe el número de itinerarios de cada clase a través del método *tratarFicheroXML(rutaPlan, rutaImagen)* y el momento en el que el

comercial del Sistema AFABLE pulsa el botón *ACEPTAR*. En el siguiente diagrama se recogen todas las posibles opciones que puede manejar el usuario:

**Figura 4-9: Diagrama de secuencia de segundo nivel**



El diagrama de secuencia se resume en los siguientes pasos:

1. El comercial del Sistema AFABLE (actor principal) introduce en el formulario el **número de despliegue**. Dicho número tiene sólo carácter informativo, y será el número en el que se desplegará el plan en el nodo correspondiente de la Red Inteligente.
2. A continuación se elegirá, marcando la opción que corresponda, el **tipo de ordenación** preferido para los diferentes itinerarios del plan. El comercial del

Sistema AFABLE, en función de los deseos del titular del Servicio 90X, podrá elegir una ordenación por longitud o una ordenación por importancia. En el siguiente capítulo se analizará la manera en la que se consiguen los dos tipos de ordenación.

3. Una vez completados los primeros dos parámetros, que son obligatorios, se procede a seleccionar las opciones personalizadas del documento. Para ello, el titular del Servicio 90X comunicará al comercial las opciones que desee de entre las siguientes:
  - **Ver TODOS los itinerarios del plan.** Al elegir esta opción quedarán deshabilitadas todas las demás.
  - **Ver itinerario de MAYOR importancia / longitud.** Se creará una sección específica con el análisis de este itinerario.
  - **Ver TODOS los itinerarios de un TIPO concreto.** Se creará una sección por cada clase de itinerarios. Quedará deshabilitada la opción de elegir itinerarios concretos.
  - **Ver sólo ALGUNOS itinerarios concretos de cada TIPO.** Se creará una sección por cada clase de itinerarios sólo con los itinerarios elegidos por el usuario. El comercial del Sistema AFABLE, deberá introducir en un campo de tipo JTextField el número de los itinerarios que desea que figuren en la documentación separados por comas. Esta opción deshabilita la posibilidad de ver todos los itinerarios de un tipo.
4. Una vez que se completan las opciones, el comercial del Sistema AFABLE pulsará el botón *ACEPTAR* para generar la documentación. Comenzará en este punto un proceso de **validación de datos** que comprobará los siguientes aspectos:
  - Existe número de despliegue y que está compuesto por nueve dígitos.
  - Se ha marcado un tipo de ordenación de las dos posibles.

- Los itinerarios concretos, si existen, se han introducido correctamente, es decir, que son dígitos separados por comas que están dentro de los rangos de cada tipo de itinerarios.



---

## 5 DISEÑO DEL SISTEMA

### 5.1 INTRODUCCIÓN

En el siguiente apartado se va a abordar las diferentes fases de diseño de la aplicación. Para ello, en primer lugar, se hará una descripción de la estructura de clases y paquetes de la que consta la herramienta y se planteará un ejemplo concreto de Plan de Abonado, a partir del cuál se podrá ver, paso a paso, cómo aplica cada una de las fases de diseño.

### 5.2 LÓGICA DE LA APLICACIÓN

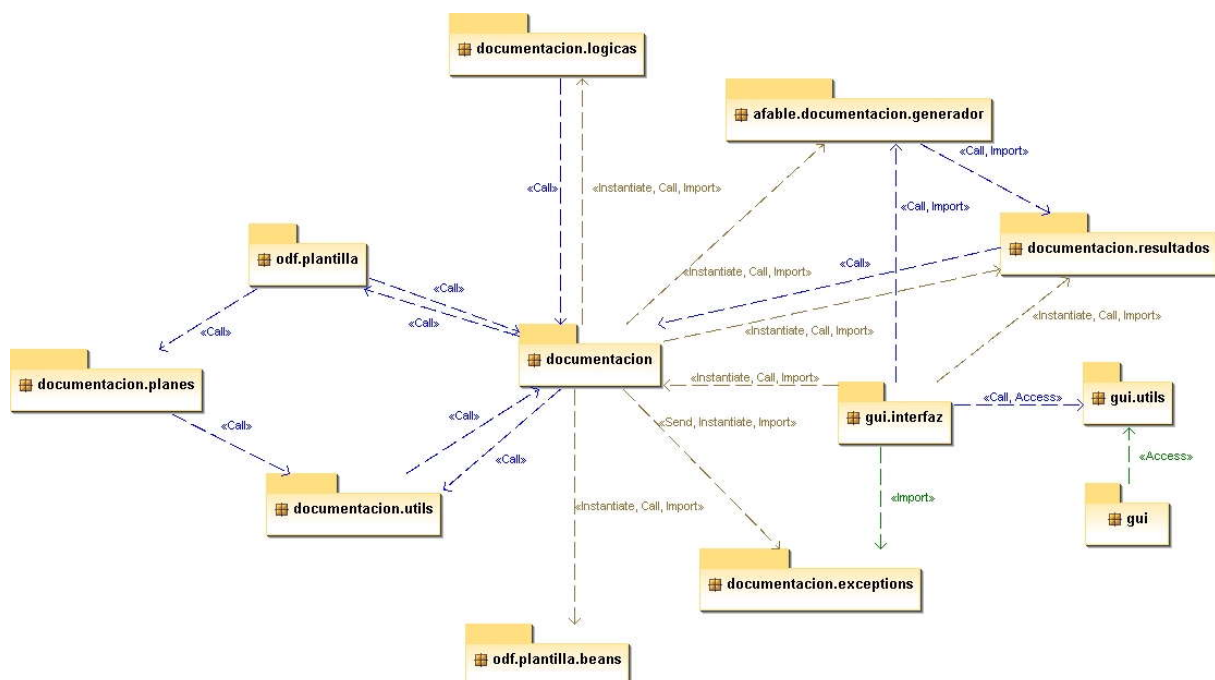
La aplicación está diseñada utilizando el paradigma de la programación orientada a objetos. Las clases se agrupan en los siguientes paquetes:

- **(default package):** Es el paquete por defecto. Contiene la clase con el método main, a partir de la cuál arranca la aplicación de forma autónoma.
- **afable.documentacion.generador:** Contiene las clases necesarias para realizar el mapeo del fichero xml del Plan de Abonado a estructuras de datos.
- **documentacion:** Contiene las clases principales para la generación de documentación.
- **documentacion.exceptions:** Contiene la clase con la que se capturarán las excepciones del sistema.
- **documentacion.logicas:** Agrupa las clases que recogen toda la lógica específica de cada una de las diferentes fases de diseño.
- **documentacion.planes:** Contiene clases que permiten generar documentos xml desde código fuente.
- **documentacion.resultados:** Presenta las distintas clases que surgen como resultados finales o intermedios de la aplicación.

- **documentacion.utils**: Agrupa distintas clases de utilidades relacionadas con la generación de la documentación.
- **gui**: Paquete base con las clases destinadas a recubrir la interfaz gráfica de la aplicación.
- **gui.interfaz**: Paquete que contiene la clase de la interfaz gráfica.
- **gui.utils**: Agrupa clases de utilidades y constantes necesarias para la lógica de la interfaz gráfica.
- **odf.plantilla**: Presenta las clases que se utilizan para la generación del documento odf de la aplicación
- **odf.plantilla.beans**: Contiene las clases que sirven como enlaces entre la documentación generada y el documento odt que se desea crear.

En la siguiente figura se muestra la relación entre los distintos paquetes de la aplicación:

**Figura 5-1: Estructura lógica de paquetes**

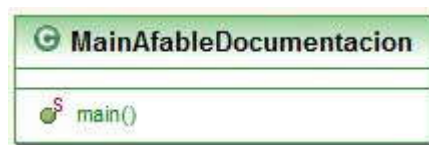


### 5.2.1 Paquete (*default-package*)

Es el paquete por defecto. Contiene una única clase con el método main que sirve para arrancar la herramienta cuando ésta funciona de forma independiente al resto de módulos del Sistema AFABLE.

- o **MainAfableDocumentacion**

**Figura 5-2: Clases paquete (*default-package*)**



### 5.2.2 Paquete *afable.documentacion.generador*

Contiene las clases necesarias para el mapeo de documentos xml a estructuras de datos. Las clases son las siguientes:

- o **Constantes**

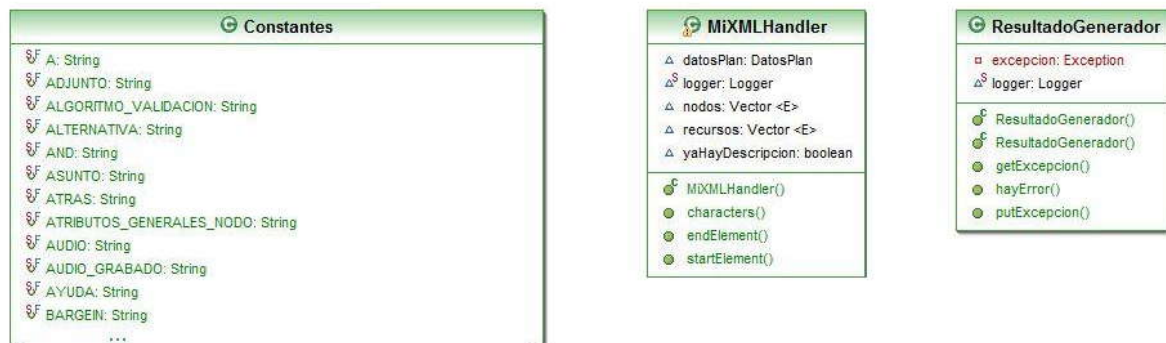
Contiene todas las constantes necesarias para el mapeo del fichero xml del Plan de Abonado a objetos.

- o **MiXMLHandler**

Realiza la lógica del mapeo, guardando cada uno de los Módulos de Lógica del Plan de Abonado en dos posiciones consecutivas de un vector. La primera de ellas reservada al tipo de nodo y la segunda al objeto en sí del nodo. Del mismo modo se recogen también los recursos del plan. Se verá más en detalle en la primera fase del diseño.

- o **ResultadoGenerador**

Recoge el resultado de la operación de mapeo. Si se produce alguna excepción, queda recogida en este objeto.

**Figura 5-3: Clases paquete *afable.documentacion.generador***

### 5.2.3 Paquete *documentacion*

Recoge las clases más importantes de la aplicación. Están relacionadas con la ordenación y clasificación de los distintos itinerarios. Las clases que componen este paquete son las siguientes:

#### o **AfableDocumentacion**

Clase principal de la herramienta de documentación. Será con la que arranque la aplicación tanto si funciona de forma autónoma como si está integrada dentro del AFABLE IDE. A partir de aquí todo es común. Realiza dos acciones:

- Inicia los parámetros de configuración.
- Arranca el tratamiento del fichero xml del plan.

#### o **IndicesPorLongitud**

Clase que recoge los índices de los itinerarios ordenados por longitud (número de nodos). Permite ordenar rápidamente los distintos itinerarios en función de su longitud.

#### o **IndicesPorPeso**

Clase que recoge los índices de los itinerarios ordenados por importancia (peso de los nodos). Permite ordenar rápidamente los distintos itinerarios en función de su importancia.

o **InformacionNodo**

Clase que recoge la información más importante del nodo, como es su nombre, tipo, descripción, posición y peso.

o **ItinerarioCompleto**

Objeto que recoge un itinerario completo del Plan de Abonado. Los itinerarios se representan con objetos cuya estructura interna tiene forma de árbol.

o **ItinerariosAzules**

Contiene todos los itinerarios sin salida válida del plan. Dispone de tres vectores de objetos cuya estructura interna tiene forma de árbol, uno con la posición original de los itinerarios, otro, con los itinerarios ordenados por longitud y un tercero con los itinerarios ordenados por importancia. Mantiene información también del número de itinerarios totales de este tipo que existen en el Plan de Abonado.

o **ItinerariosNegros**

Misma estructura lógica que en el caso anterior pero para los itinerarios correctos del Plan de Abonado.

o **ItinerariosRojos**

Misma estructura lógica que en el caso anterior pero para los itinerarios con errores del Plan de Abonado.

o **ItinerariosVerdes**

Igual que en los tres casos anteriores pero para los itinerarios con eventos de marcha atrás en el Plan de Abonado.

o **LectorPlanXML**

Es el corazón de la herramienta de documentación. Clase utilizada para leer un plan XML a partir de su ruta y opcionalmente una imagen del plan. Realiza el tratamiento del fichero del plan. Establece y clasifica todos los posibles itinerarios. Analiza también todos los recursos del plan. Con toda la información genera un documento en el estándar libre de OpenOffice (.odt) y lo aloja en la misma ruta del plan.

o **MatrizSaltos**

Clase que guarda la información de los siguientes saltos de todos los nodos en el Plan de Abonado. Será la clase protagonista en el algoritmo de ordenación de nodos.

o **NodoArbol**

Clase que representa el objeto identificativo de cada nodo en el Plan de Abonado, con el que se irá construyendo el árbol asociado al Plan.

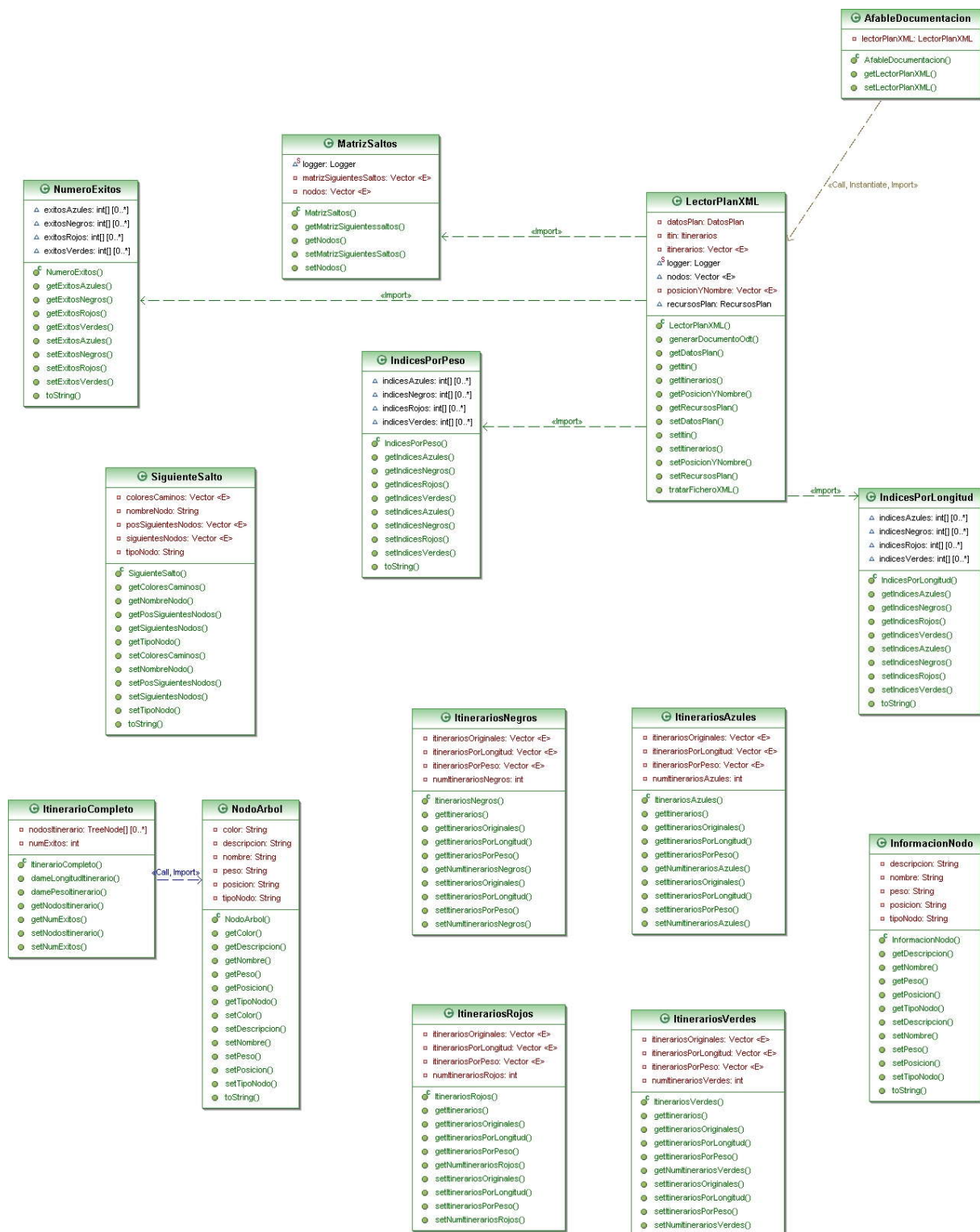
o **NumeroExitos**

Clase que recoge el número de “éxitos” que presenta cada itinerario. Más adelante, en la segunda fase de diseño se detallará que se entiende por éxito en un itinerario de un Plan de Abonado.

o **SiguienteSalto**

Clase que representa las diferentes opciones que puede cursar una llamada en un Plan, a partir de cada uno de los nodos.

El diagrama de clases se recoge en la siguiente figura:

Figura 5-4: Clases paquete *documentacion*

#### 5.2.4 Paquete *documentacion.exceptions*

Contiene la clase encargada de gestionar el control de excepciones.

- o **AfableDocException**: Si se produce cualquier error en la generación de la documentación automática, se provoca una excepción modelada con un objeto de esta clase.

**Figura 5-5: Clases paquete *documentacion.exceptions***



#### 5.2.5 Paquete *documentacion.logicas*

Agrupar las clases que aglutinan y recubren, a alto nivel, la lógica asociada a cada una de las fases de diseño de la aplicación. Las clases son las siguientes:

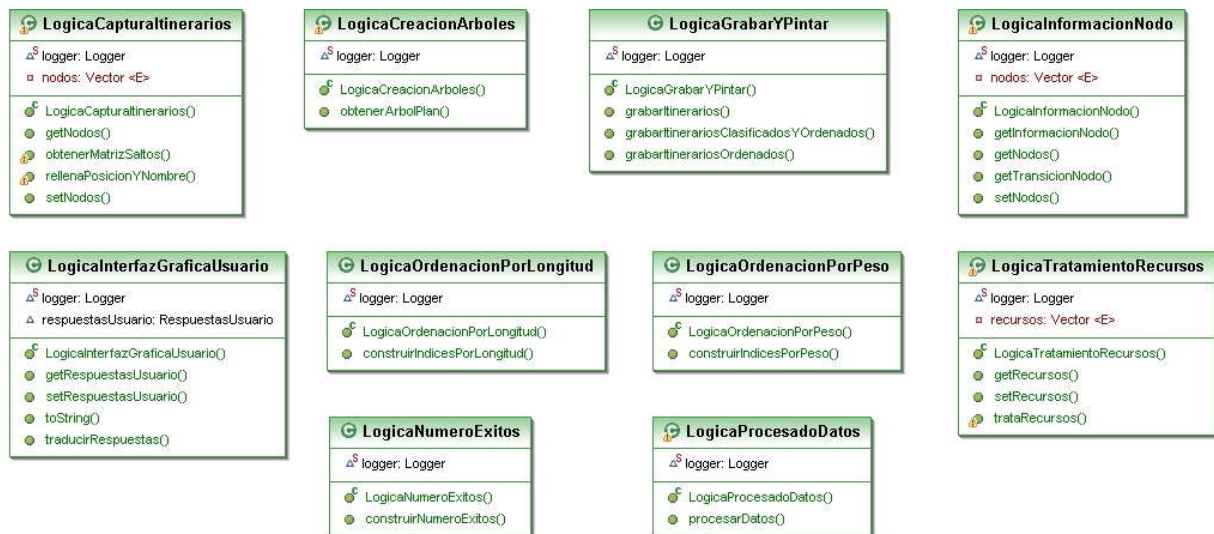
- o **LogicaCapturaItinerarios**: Recubre las funciones que se encargan de implementar la primera parte del algoritmo de ordenación de nodos. Básicamente, recoge, para cada nodo del Plan de Abonado, los posibles siguientes nodos a los que puede acceder la llamada y el color de sus transiciones.
- o **LogicaCreacionArboles**: Recoge las funciones que se encargan de gestionar la segunda parte del algoritmo de ordenación de nodos y todo lo referente a la detección de bucles en los itinerarios del Plan de Abonado. Básicamente, crea por cada Plan un árbol donde cada rama completa representa un itinerario válido del Plan.
- o **LogicaGrabarYPintar**: Aglutina las funciones que se encargan de grabar y pintar los posibles itinerarios en ficheros de texto. Los ficheros de texto con listados de los itinerarios clasificados y ordenados tienen una doble misión. Por un lado, representan un papel de trazas esencial en la depuración de los



algoritmos y por otro, permiten al usuario visualizar rápidamente itinerarios de interés a incluir en la documentación automática.

- o **LogicaInformacionNodo:** Recoge todas las funciones que permiten recuperar la información más importante de cada uno de los Módulos de Lógica que intervienen en el Plan de Abonado. Habrá una función específica por cada Módulo de Lógica.
- o **LogicaInterfazGraficaUsuario:** Recoge la lógica relacionada con las opciones que ha elegido el usuario en la interfaz gráfica para generar la documentación automática del Plan.
- o **LogicaNumeroExitos:** Recoge la funcionalidad relacionada con el número de “éxitos” que tiene cada itinerario. Más adelante se especificará en que consisten los “éxitos” de un itinerario y que influencia tienen en el nivel de importancia de los mismos.
- o **LogicaOrdenacionPorLongitud:** Contiene las funciones necesarias para la ordenación de los diferentes itinerarios de un Plan de Abonado en función de su longitud, esto es, del número de nodos que conforma cada itinerario.
- o **LogicaOrdenacionPorPeso:** Contiene las funciones necesarias para la ordenación de los diferentes itinerarios de un Plan de Abonado en función de su importancia, esto es, del peso de cada itinerario. El peso de un itinerario se calculará como la suma de los pesos de todos los nodos que componen el itinerario como quedará explicado en la tercera fase de diseño.
- o **LogicaProcesadoDatos:** Aglutina las funciones y objetos necesarios productos del procesamiento de todos los datos conseguidos hasta el momento por la aplicación de cara a preparar la herramienta para la generación del documento automático en formato odf.
- o **LógicaTratamientoRecursos:** Contiene las funciones relacionadas con el tratamiento de los recursos del Plan de Abonado.

El diagrama de clases se presenta en la siguiente figura:

**Figura 5-6: Clases paquete *documentacion.logicas***

### 5.2.6 Paquete *documentacion.planes*

Este paquete contiene una única clase destinada a manipular ficheros xml desde código fuente. Resulta útil para representar la información de scripts en el documento automático del Plan de Abonado.

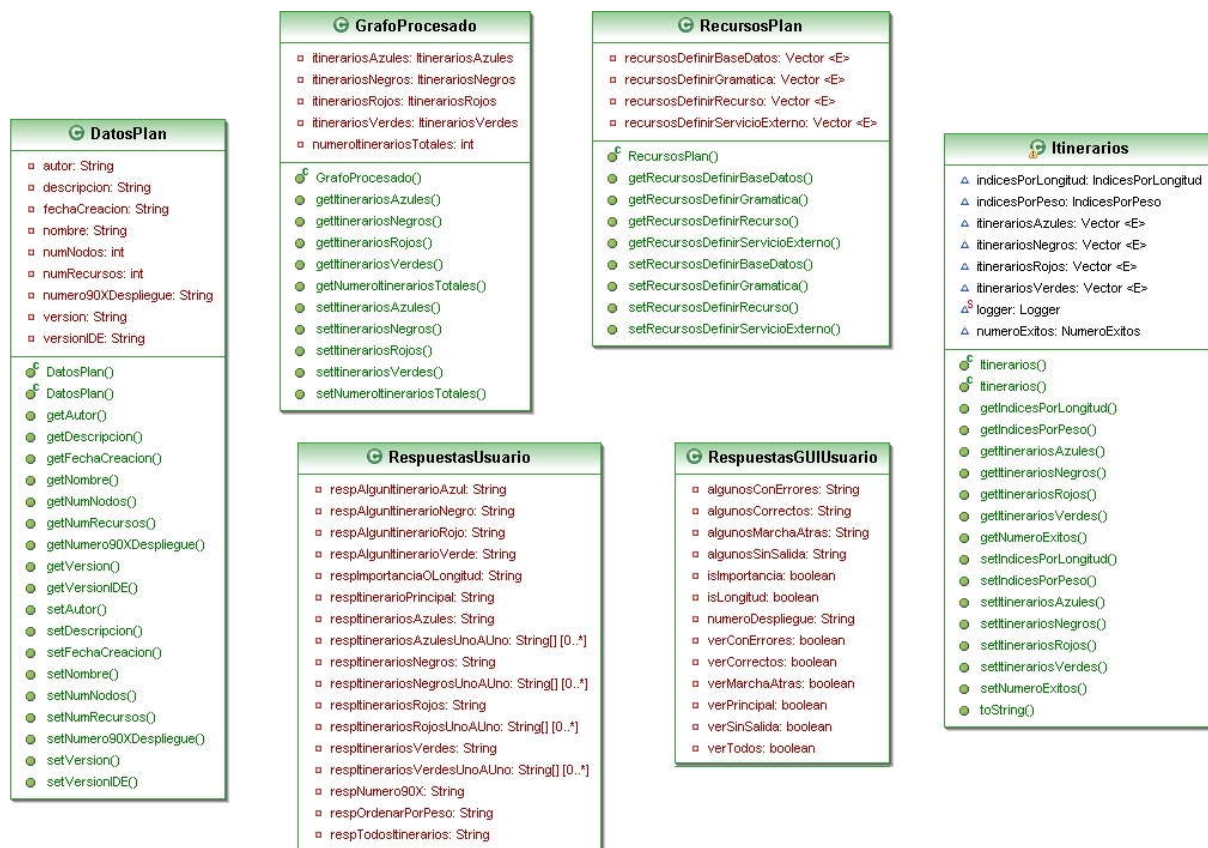
- o **GeneraDocumentoXML**: Contiene funciones para crear, modificar y borrar ficheros xml o porciones de código xml desde código fuente.

**Figura 5-7: Clases paquete *documentacion.planes***

### 5.2.7 Paquete *documentacion.resultados*

Este paquete contiene las clases obtenidas como resultados finales o parciales de las diferentes fases de diseño. Son las siguientes:

- o **DatosPlan**: Recoge la información general de un Plan de Abonado. Sirve para crear la sección del documento automático del mismo nombre. Contendrá datos relacionados con el nombre del plan, su versión, el número de nodos, número de itinerarios de cada tipo, número de recursos, etc.
- o **GrafoProcesado**: Encapsula a más alto nivel los cuatro tipos de itinerarios que pueden existir en un Plan de Abonado.
- o **Itinerarios**: Objeto que modela los itinerarios del Plan de Abonado. Contendrá un vector de itinerarios por cada uno de los cuatro tipos que pueden existir, además de la información necesaria para poder ordenarlos en cualquier momento según los dos criterios de ordenación establecidos para la aplicación.
- o **RecursosPlan**: Objeto que modela los recursos existentes en el Plan de Abonado. Dispondrá de un vector por cada uno de los cuatro tipos de recursos que pueden existir.
- o **RespuestasGUIUsuario**: Clase que recoge las opciones elegidas por el usuario en la interfaz gráfica para la generación del documento automático del plan.
- o **RespuestasUsuario**: Clase original ideada para recoger las respuestas del usuario en la primera versión de la aplicación. En dicha versión, se planteaba una interfaz textual basada en línea de comandos. Con el fin de reutilizar código y para no eliminar la posibilidad de poder utilizar en un futuro esta opción, lo que se hace es una traducción de las respuestas del usuario en la interfaz gráfica a este objeto.

Figura 5-8: Clases paquete *documentacion.resultados*

## 5.2.8 Paquete *documentacion.utilis*

Este paquete contiene las clases destinadas a leer los parámetros de los diferentes ficheros de configuración, así como alguna que otra clase de utilidad para la aplicación. Dichos ficheros estarán escritos en lenguaje xml, de modo que cualquier cambio de configuración se pueda realizar sin tener que modificar los ficheros fuente. Están ubicados en la ruta */DocumentacionAutomatica/Recursos*. Las clases que componen el paquete son las siguientes:

- o **Configuracion:** Lee los parámetros de configuración del fichero *configuracion.xml*. Básicamente, parámetros relacionados con las rutas de los diferentes recursos y del fichero de trazas.

- o **Iconos:** Lee los parámetros relacionados con las imágenes de los distintos iconos que intervienen en la aplicación. El fichero *iconos.xml*, pondrá una etiqueta a cada uno de los iconos de la aplicación.
- o **Literales:** Lee las constantes contenidas en el fichero *constantes.xml*.
- o **Mensajes:** Lee los mensajes de usuario utilizados en la aplicación y que están contenidos en el fichero *mensajes.xml*.
- o **Preguntas:** Lee las diferentes opciones contenidas en el fichero *preguntas.xml* y que servirán para generar la interfaz gráfica de usuario.
- o **Secciones:** Lee del fichero *secciones.xml*, los títulos y encabezados automáticos que se utilizan en el documento del plan que genera la aplicación.
- o **Trazas:** Clase que contiene una serie de métodos estáticos que facilitan la tarea de generar trazas para objetos complejos de la aplicación.

**Figura 5-9: Clases paquete *documentacion.utils***



### 5.2.9 Paquete *gui*

Paquete que contiene una única clase que presenta el entorno gráfico de recogida de parámetros.

- o **AfableGUIDocumentacion**: Clase que presenta el entorno gráfico en el que se recogen los parámetros, estos son, la ruta del fichero xml del plan y la ruta del fichero *.jpg* ó *.png* que contiene la imagen del plan. Esta clase es exclusiva del funcionamiento autónomo de la herramienta.

**Figura 5-10: Clases *paquete gui***



### 5.2.10 Paquete *gui.interfaz*

Paquete que contiene la clase básica para el modelado de la interfaz gráfica con la que se solicita al usuario las opciones con las que desea generar el documento automático del Plan de Abonado.

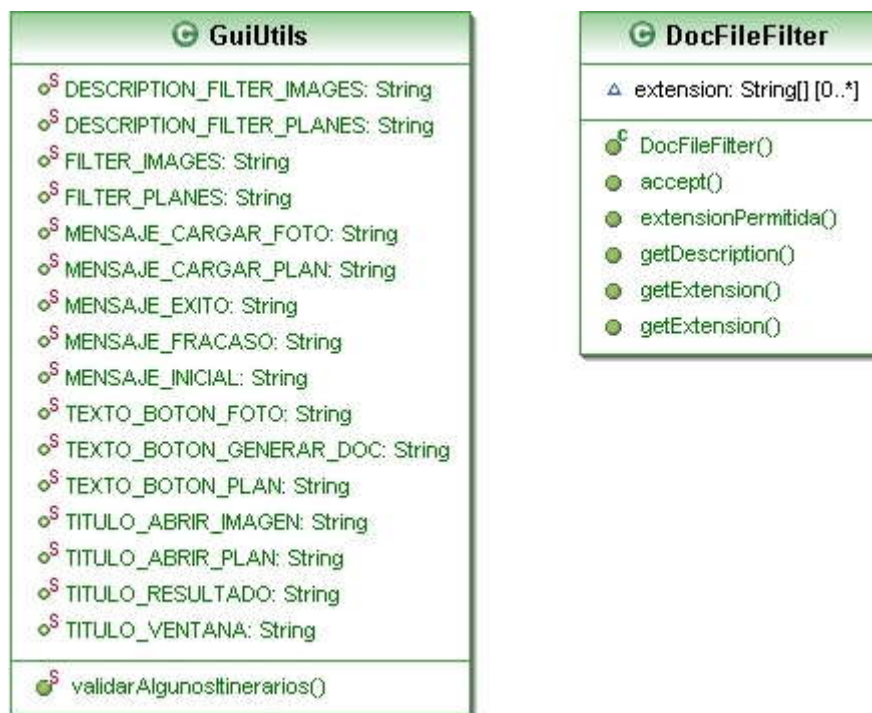
- o **InterfazGraficaUsuario**: Contiene el frame con el interfaz gráfico con las opciones que permiten al usuario personalizar el documento automático generado por la aplicación.

**Figura 5-11: Clases *paquete gui.interfaz***

### 5.2.11 Paquete *gui.utils*

Paquete que contiene una clase de utilidades para la interfaz gráfica.

- o **GuiUtils**: Contiene constantes relacionadas con títulos de ventanas, nombre de etiquetas, etc. Además provee métodos de validación de las respuestas elegidas por el usuario.
- o **DocFileFilter**: Clase que modela un filtro para las ventanas de selección de ficheros. Para la selección de un Plan de Abonado filtra todos los ficheros de extensión *.xml*, mientras que para la selección de la imagen del Plan filtra los ficheros de extensiones *.jpg* y *.png*.

**Figura 5-12: Clases paquete gui.utils**

### 5.2.12 Paquete *odf.plantilla*

Paquete que contiene todas las clases relacionadas con la generación del documento automático del Plan de Abonado en el estándar libre OpenDocument (odf). Las clases son las siguientes:

- o **Estilos**: Clase que recoge las propiedades de todos y cada uno de los estilos generados de forma exclusiva para el documento de la aplicación.
- o **EstilosUtil**: Clase de apoyo que contiene un listado de los nombres de todos los estilos.
- o **Plantilla**: Es la clase más importante de la aplicación en lo que se refiere a la creación del documento .odt. Se encarga de crear el documento e ir rellenando cada una de las secciones del mismo en base a los datos recogidos en el objeto DatosProcesadosDocumentoODF. En las fases VIII y IX, se

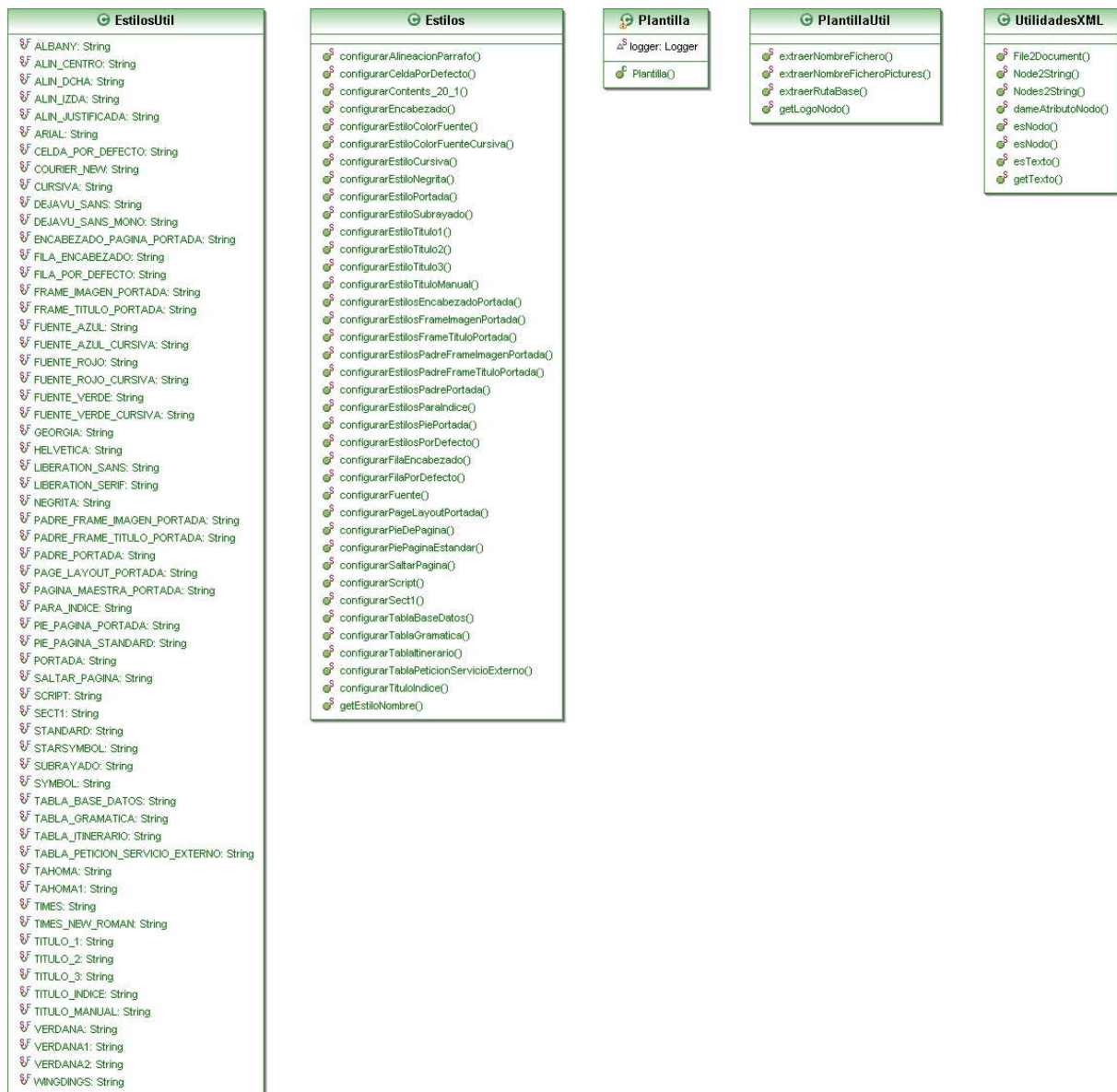


explica detalladamente todos los aspectos relacionados con la forma y el contenido del fichero generado.

- o **PlantillaUtil**: Clase de apoyo que contiene métodos útiles para agilizar la lógica de la generación del documento.
- o **UtilidadesXML**: Contiene funciones para representar código xml en el documento .odt del Plan de Abonado, de forma que se mantenga el sangrado de etiquetas.

A continuación se muestra el diagrama de clases del paquete:

Figura 5-13: Clases paquete odf.plantilla



### 5.2.13 Paquete *odf.plantilla.beans*

Paquete que contiene la clase que va a servir de enlace entre la primera parte de obtención y clasificación de itinerarios y esta segunda parte de creación del documento .odt del Plan de Abonado.

- o **DatosProcesadoDocumentoODF**: Contiene toda la información necesaria para poder construir un documento del Plan de Abonado en base a las especificaciones requeridas. Dispone, entre otros, de los siguientes objetos:
  - **GrafoProcesado**: Objeto con la información de los cuatro tipos de itinerarios.
  - **DatosPlan**: Objeto con los datos generales del Plan de Abonado.
  - **RespuestasUsuario**: Objeto con las opciones elegidas por el usuario para personalizar el documento de su plan.
  - **RecursosPlan**: Objeto con los recursos del plan.

**Figura 5-14: Clases paquete odf.plantilla.beans**

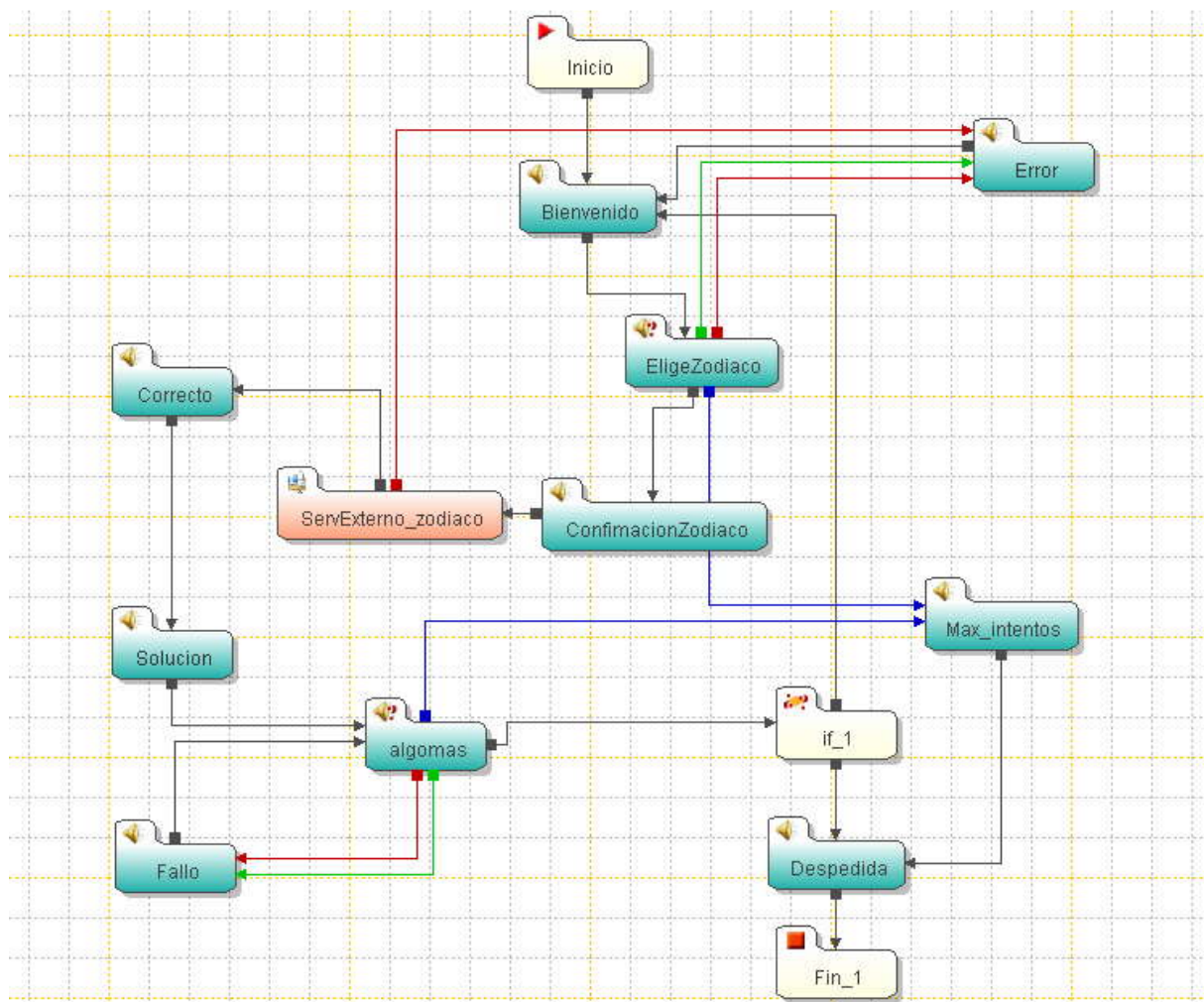


### 5.3 FASES DE DISEÑO

En el presente apartado se van a explicar detalladamente cada una de las fases de diseño de las que consta la aplicación. Con el objetivo de ilustrar cada una de las etapas de diseño se va a plantear, a modo de ejemplo, un Plan de Abonado sencillo, de forma que se pueda ver de modo práctico cómo aplica el algoritmo de ordenación de nodos y cómo se genera el documento automático del Plan.

#### **Plan de Abonado: “*ConsultaHoroscopo.xml*”**

El Plan de Abonado *consultaHoroscopo.xml* implementa un Servicio 90X de consulta de signos del zodiaco. Permite a los llamantes obtener información de un signo del zodiaco. El grafo del Plan de Abonado se detalla a continuación:

**Figura 5-15: Plan de Abonado “consultaHoroscopo.xml”**

Como se puede ver en el grafo del plan, una llamada al Servicio 90X se encontrará en primer término con una locución de bienvenida, a la que seguirá una nueva locución que pedirá al llamante el signo del zodiaco sobre el que quiere realizar la consulta. Posteriormente, tras la confirmación del signo, le devolverá la información a partir de la invocación a un servicio externo. Antes de finalizar la llamada, si todo ha ido bien, se preguntará al usuario si desea información de algún otro signo, en cuyo caso se repetirá el proceso anterior.

### 5.3.1 FASE I: Mapeo del fichero *.axml* a estructuras de datos

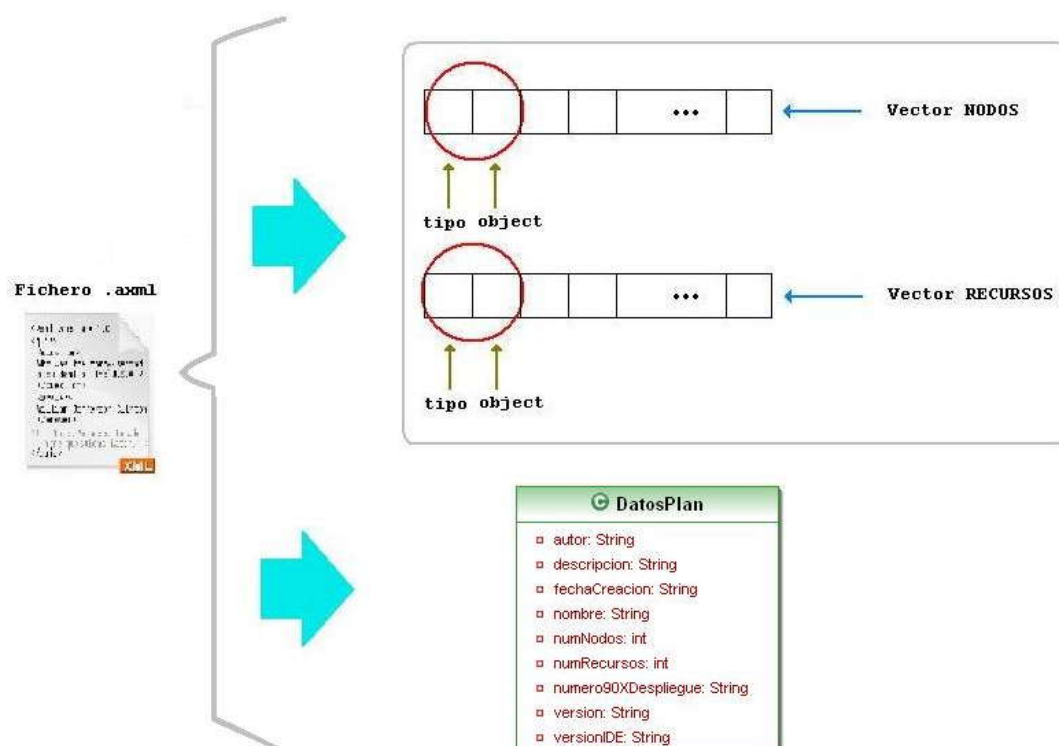
La primera fase de diseño consiste en traducir el fichero *.axml* del Plan de Abonado a objetos, de forma que se pueda trabajar con ellos para el resto de tareas de la aplicación. Para ello, se hace uso de un parser XML.

Cada uno de los nodos del Plan de Abonado tiene una representación como objeto, de forma que los atributos y etiquetas hijas del código xml, se convierten en propiedades y métodos del objeto. Los distintos Módulos de Lógica y la información más importantes que se utilizará para elaborar el documento automático se detallan en el apartado **4.3 DESCRIPCIÓN DE LOS MÓDULOS DE LÓGICA**.

El objetivo, en esta primera fase de diseño, es procesar el código xml para obtener dos vectores de datos. El primer vector será el vector general de nodos y contendrá todos los objetos representativos de los Módulos de Lógica del Plan. El segundo vector será el vector general de recursos y contendrá todos los objetos característicos de los diferentes Recursos del Plan. Además se recogerán los datos generales del Plan de Abonado, como son su nombre, descripción, versión, etc. y se empezará a construir el objeto **DatosPlan** cuya misión es contener los datos generales del Servicio 90X para crear su correspondiente sección en el documento automático generado por la aplicación.

En la siguiente figura se ilustra lo que se pretende conseguir con esta primera fase de diseño:

Figura 5-16: Mapeo del fichero .axml a estructuras de datos



La información del vector *posicionYNombre* para el plan **consultaHoroscopo.axml** se recoge en la siguiente tabla:

Tabla 5-1: Vector *posicionYNombre* del Plan **consultaHoroscopo.axml**

POSICIÓN	NOMBRE	PESO	DESCRIPCIÓN
0	<b>nodo_inicio</b>	0	Nodo inicio del plan
2	<b>Error</b>	0	Locución de error
4	<b>Fallo</b>	0	Fallo en el dato de entrada
6	<b>Fin_1</b>	0	Nodo fin
8	<b>Despedida</b>	0	Locución de despedida
10	<b>Bienvenido</b>	0	Locución de bienvenida
12	<b>Max_intentos</b>	0	Límite de intentos

14	<b>if_1</b>	0	Nodo if
16	<b>ConfirmacionZodiaco</b>	0	Confirma el signo del zodiaco
18	<b>Correcto</b>	0	Locución todo correcto
20	<b>Solucion</b>	0	Locución con la predicción
22	<b>EligeZodiaco</b>	0	Locución para elegir signo
24	<b>ServExterno_zodiaco</b>	1	Servicio externo que realiza la consulta
26	<b>algotmas</b>	0	Locución para más signos

### 5.3.2 FASE II: Algoritmo de Ordenación de Nodos

En la Fase II, se plantea la forma en la que procesar la información de los dos vectores de datos resultados de la primera fase, de forma que se puedan elaborar los diferentes itinerarios que puede seguir una llamada al Servicio 90X.

El hecho fundamental que va a permitir llegar hasta este punto es la información que posee cada uno de los Módulos de Lógica sobre los siguientes nodos a los que se puede acceder en la siguiente transición.

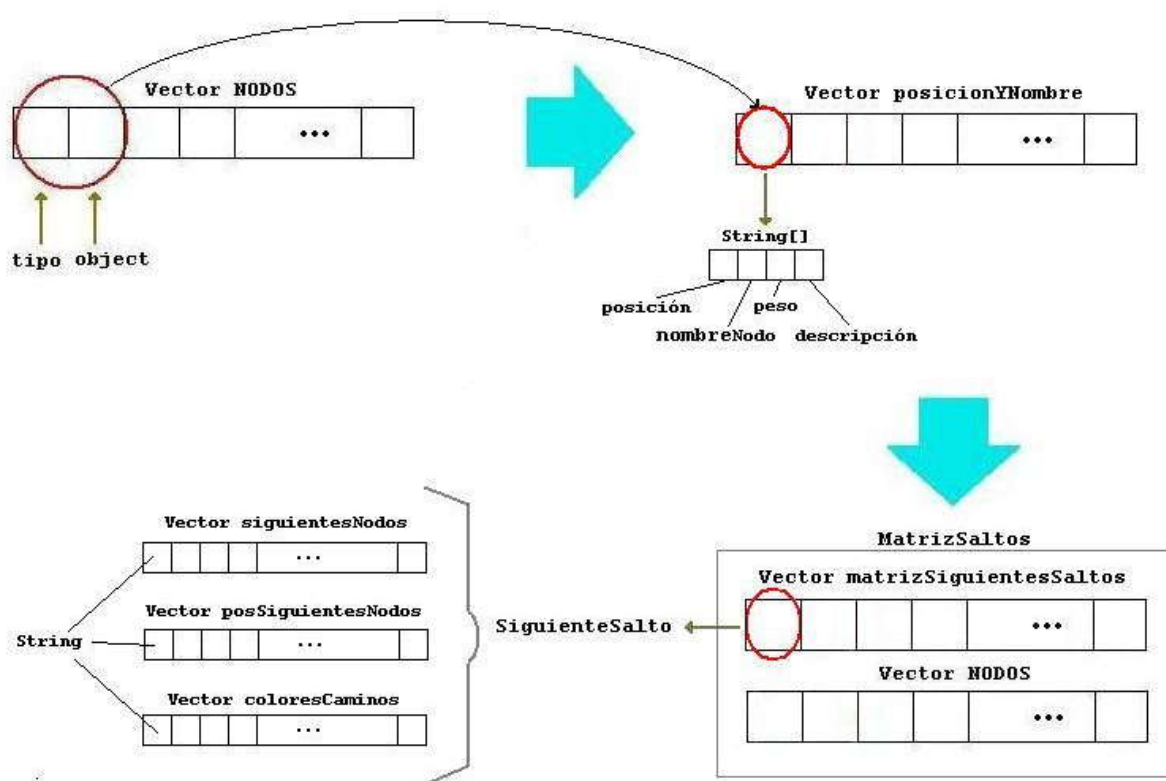
El **Algoritmo de Ordenación de Nodos**, se puede resumir en los siguientes pasos:

1. Se analiza el vector general de nodos y se crea un nuevo vector auxiliar, denominado *posicionYNombre*, compuesto por arrays de cadenas de cuatro elementos:
  - Posición del nodo en el vector general de nodos.
  - Nombre del nodo.
  - Peso del nodo. Más adelante se detallará como se calcula el peso de un nodo.
  - Descripción del nodo.



- Se establece para cada nodo, todos los posibles siguientes nodos que puede alcanzar en el siguiente salto, guardando memoria del color de la transición con la que se llega a cada uno de los nodos siguientes. De esta forma se recoge toda la información en un objeto de tipo **MatrizSaltos**. El mapeo de la información en los distintos objetos se recoge en la siguiente imagen:

**Figura 5-17: Algoritmo de Ordenación de Nodos. Mapeo de objetos**



La información esencial del objeto **MatrizSaltos** aplicada al Plan de Abonado *consultaHoroscopo.xml* se recoge en la siguiente tabla. Por cada nodo, se puede ver el nombre de los nodos a los que se puede acceder en el siguiente salto, la posición que ocupan dichos nodos en el vector general de nodos y el color de la transición que se lleva a cabo con el salto.

**Tabla 5-2: Matriz de Saltos para el Plan *consultaHoroscopo.xml***

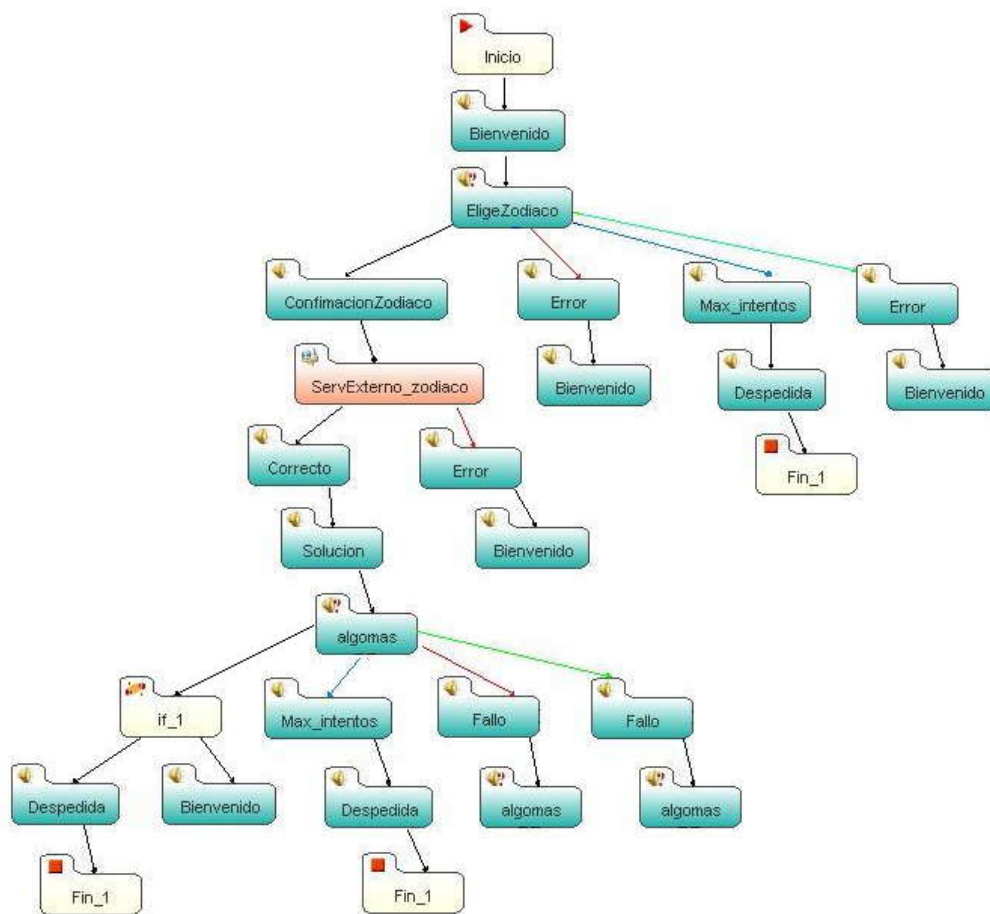
NOMBRE	SALTOS	POSICIÓN	COLOR
nodo_inicio	[Bienvenido]	[10]	[negro]
Error	[Bienvenido]	[10]	[negro]
Fallo	[algomas]	[26]	[negro]
Fin_1	[]	[]	[]
Despedida	[Fin_1]	[6]	[negro]
Bienvenido	[EligeZodiaco]	[22]	[negro]
Max_intentos	[Despedida]	[8]	[negro]
if_1	[Bienvenido, Despedida]	[10, 8]	[negro, negro]
ConfirmacionZodiaco	[ServExterno_zodiaco]	[24]	[negro]
Correcto	[Solucion]	[20]	[negro]
Solucion	[algomas]	[26]	[negro]
EligeZodiaco	[ConfirmacionZodiaco, Error, Max_intentos, Error]	[16, 2, 12, 2]	[negro, verde, azul, rojo]
ServExterno_zodiaco	[Correcto, Error]	[18, 2]	[negro, rojo]
algomas	[if_1, Fallo, Max_intentos, Fallo]	[14, 4, 12, 4]	[negro, verde, azul, rojo]

3. Se crea una estructura de datos que modele el Plan de Abonado en función de la información contenida en el objeto **MatrizSaltos**. Dicha estructura de datos será un *árbol*. Las ramas del árbol serán los itinerarios que deseamos conseguir. La construcción del árbol realiza los siguientes pasos:
  - Crea el primer nodo del árbol a partir del Nodo de Inicio que es común para todos los planes.

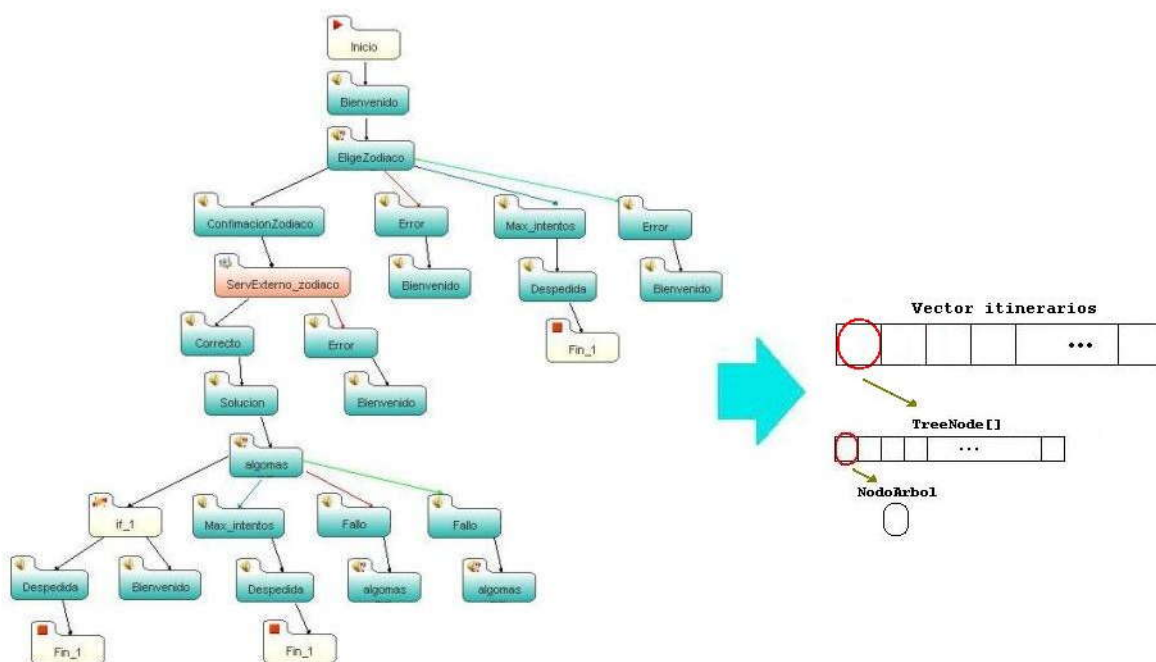
- Se construye el árbol insertándole como nodo raíz el primer nodo.
- Se obtiene el objeto **SiguienteSalto** de la matriz de saltos correspondiente al primer nodo.
- Se llama a un método que inserta como nodos hijos en el árbol todos los posibles siguientes nodos contemplados en el objeto **SiguienteSalto**.
- Se procede a repetir la misma operación de forma recursiva hasta que se encuentre un nodo que no tenga definidos más saltos, como puede ser el Nodo Fin.
- **Técnicas “Antibucles”**: Se entiende por Técnicas Antibucles el conjunto de mecanismos llevados a cabo para asegurar que la tarea recursiva que inserta los nodos en el árbol del Plan de Abonado termina siempre.

Puede ocurrir que una llamada al Servicio 90X realice una transición hacia un Módulo de Lógica por el que ya ha pasado previamente. En ese caso, el algoritmo detecta que el nuevo nodo a insertar en el árbol está ya repetido en esa rama y procede a finalizar el itinerario. Para ello, anula los siguientes saltos del nodo repetido.

En la siguiente figura se muestra el árbol del Plan *consultaHoroscopo.axml* resultado de la aplicación del **Algoritmo de Ordenación de Nodos**:

**Figura 5-18: Árbol del Plan consultaHoroscopo.axml**

4. Una vez construido el árbol, el siguiente paso es construir el vector general de itinerarios. Para ello, se recorre la estructura de forma recursiva desde el nodo raíz hasta cada uno de los nodos “hoja”. De esta forma, obtenemos la relación de Módulos de Lógica que componen cada una de las ramas, o lo que es lo mismo, el conjunto de nodos que conforman cada itinerario. El mapeo de caminos desde el árbol al vector de itinerarios se representa en la siguiente imagen:

**Figura 5-19: Mapeo de caminos al vector de itinerarios**

### 5.3.3 FASE III: Clasificación y ordenación de itinerarios

En la tercera fase de diseño, se pretende clasificar y ordenar los itinerarios contenidos en el vector general de itinerarios construido durante la fase anterior. Para ello, se dispone de un objeto **Itinerarios**, cuya descripción se detalla a continuación:

#### Clasificación de itinerarios

Para clasificar los itinerarios se recorre el vector general de itinerarios. Por cada itinerario se accede hasta el contenido de cada uno de los nodos. Una vez que se ha llegado al contenido de cada nodo representado por el objeto **NodoArbol**, se consulta el color de la transición con la que se ha alcanzado dicho nodo. La clasificación se realiza de la siguiente manera:

- Se crea un vector de itinerarios que contiene todos los **itinerarios correctos** del plan. Éstos serán aquellos cuyas transiciones sean todas de color

“**negro**”. Por itinerarios correctos se entiende todos aquéllos caminos que puede cursar la llamada en el Plan de Abonado de forma que transcurra de forma normal desde su inicio hasta su finalización.

- Se crea un vector de **itinerarios sin salida válida**. Éstos serán aquellos caminos en los que exista al menos un nodo cuya transición sea de color “**azul**”. Los itinerarios de este tipo se producen cuando se dispara un evento concreto en el transcurso de la llamada, por ejemplo, cuando el analizador de voz no reconoce una respuesta del usuario.
- Se crea un vector de **itinerarios con eventos de error**. Éstos serán aquellos itinerarios en los que exista al menos un nodo cuya transición sea de color “**rojo**”. Los caminos de este tipo se producen cuando surge un error inesperado en la lógica de alguno de los Módulos. Puede ocurrir, en el ejemplo que contemplamos, que falle la conexión con el servicio externo que proporciona la información del zodiaco al Plan.
- Se crea un vector de **itinerarios con eventos de marcha atrás**. Éstos serán aquellos caminos en los que exista al menos un nodo cuya transición sea de color “**verde**”. Los caminos de este tipo se producen cuando en un nodo del Plan se contempla la posibilidad de que el usuario mediante el vocablo “*ATRÁS*”, retroceda a un módulo anterior.

## Ordenación de itinerarios

Existen dos formas de ordenar los itinerarios:

- **Ordenación por Longitud**: Se atiende única y exclusivamente al número de nodos que conforma cada uno de los itinerarios. En base a ello, se ordenarán los caminos de mayor a menor número de nodos. Se gestiona a partir del objeto **IndicesPorLongitud**, que contendrá un vector de índices por cada grupo de itinerarios. De esta forma, para conocer los itinerarios de un grupo

ordenados por longitud, habría que manejar el vector original de itinerarios de ese grupo y su vector de índices asociado.

- **Ordenación por Importancia:** Es posible ordenar los itinerarios de un Plan de Abonado en función del peso que tiene cada uno de ellos. El peso de un itinerario es la suma directa de los pesos de todos los nodos que conforman dicho itinerario. Para calcular el peso de un nodo, hay que tener en cuenta los siguientes aspectos:
  - Todos los objetos que representan Módulos de Lógica contienen como atributo un objeto de tipo **AtributosGeneralesNodo**. Dicho objeto, contiene información general del módulo, como puede ser su posición en el layout de dibujo del AFABLE IDE, información de facturación, nombre, descripción, etc. Entre esta información, destacan dos atributos, que son el atributo **“trazar”** y el atributo **“éxito”**. El primero de ellos, se define para indicar que deben existir trazas de información del paso de la llamada por el nodo. El segundo de ellos, cuando se define, asigna una determinada etiqueta al nodo cuando se cursa la llamada de forma correcta por él.
  - El peso de un nodo se calcula a partir de estos dos atributos y puede tener los siguientes valores:

**Tabla 5-3: Tabla de pesos de un nodo**

Atributo <b>“trazar”</b>	Atributo <b>“éxito”</b>	<b>PESO NODO</b>
No definido	No definido	<b>0</b>
Definido	No definido	<b>1</b>
No definido	Definido	<b>5</b>
Definido	Definido	<b>6</b>

Como se puede ver en la tabla, el atributo “*trazar*” contribuye al peso de un nodo en una unidad, mientras que el atributo “*éxito*” contribuye en cinco unidades. El objetivo fundamental que se persigue con este tipo de ordenación es facilitar que un usuario pueda establecer itinerarios de mayor o menor peso configurando estos dos atributos en los nodos que desee. Con ello consigue poder generar una documentación más personalizada.

Imaginemos que un cliente está muy interesado en obtener la documentación de un itinerario concreto y que desea que ese itinerario sea, además, su itinerario principal. Puede conseguir este objetivo manipulando los atributos de los nodos del itinerario que desea. Después puede generar el documento automático del Plan seleccionando como opción “*Ver sólo el itinerario de mayor importancia*”. En ese caso conseguiría obtener un documento cuyo itinerario principal ha sido establecido por él.

La ordenación de caminos por peso se realizará también de mayor a menor y estará gestionada por el objeto **IndicesPorPeso**, que al igual que en el caso anterior, manejará vectores de índices por cada uno de los cuatro grupos de itinerarios que existen.

#### 5.3.4 FASE IV: Tratamiento de los recursos

Al igual que se ha explicado para el caso de los nodos, los recursos del Plan requieren también un tratamiento. La cuarta fase de diseño se centra en la clasificación de los recursos contenidos en el vector general de recursos. Se distinguen cuatro grupos:

- Recursos de tipo **Base de Datos**: Son aquellos que definen una Base de Datos sobre la que distintos Módulos de Lógica pueden actuar.
- Recursos de tipo **Servicio Externo**: Establecen un mecanismo de comunicación basado en objetos Petición y Respuesta mediante los cuales, a

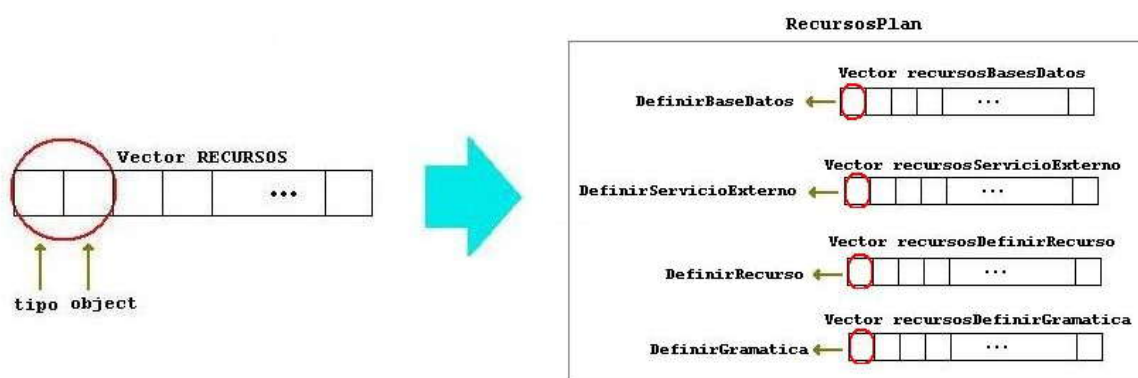


partir de una URL, posibilitan gestionar datos suministrados por agentes externos al Sistema AFABLE.

- **Ficheros de Recursos:** Son ficheros de audio, de gramáticas o rutinas ECMAScript que se introducen como recursos adjuntos al Plan de Abonado.
- Recursos de tipo **Gramática Online:** Definen gramáticas en tiempo de ejecución que pueden ser usadas por Módulos de Lógica especiales.

Para clasificar los recursos se hace uso del objeto **RecursosPlan**, tal y como se muestra en la siguiente figura:

**Figura 5-20: Clasificación de los Recursos del Plan**



### 5.3.5 FASE V: Lanzar interfaz gráfico y obtener respuestas del usuario

Una vez que se ha realizado la primera parte de análisis del Plan de Abonado y se ha aplicado el Algoritmo de Ordenación de Nodos, es necesario implementar la fase de diseño relacionada con la interfaz gráfica de usuario. Dicha fase será la responsable de personalizar el documento automático del Plan generado por la aplicación.

El listado de opciones que debe rellenar el usuario se detalla a continuación:

- **Número de despliegue** (obligatorio)
- **Tipo de ordenación** (obligatorio)

- Longitud
- Importancia
- Ver **todos** los itinerarios del plan (opcional)
- Ver el itinerario de **mayor importancia / longitud** (opcional)
- Ver **todos** los itinerarios **correctos** del plan (opcional)
- Ver **todos** los itinerarios **sin salida válida** del plan (opcional)
- Ver **todos** los itinerarios **con errores** del plan (opcional)
- Ver **todos** los itinerarios **con eventos de marcha atrás** en el plan (opcional)
- Ver sólo **algunos** itinerarios **correctos** del plan (opcional)
- Ver sólo **algunos** itinerarios **sin salida válida** del plan (opcional)
- Ver sólo **algunos** itinerarios **con errores** del plan (opcional)
- Ver sólo **algunos** itinerarios **con eventos de marcha atrás** en el plan (opcional)

Se puede encontrar información más detallada de todas estas opciones en el Manual de Usuario de la aplicación **B.2 INTERFAZ GRÁFICA DE USUARIO**.

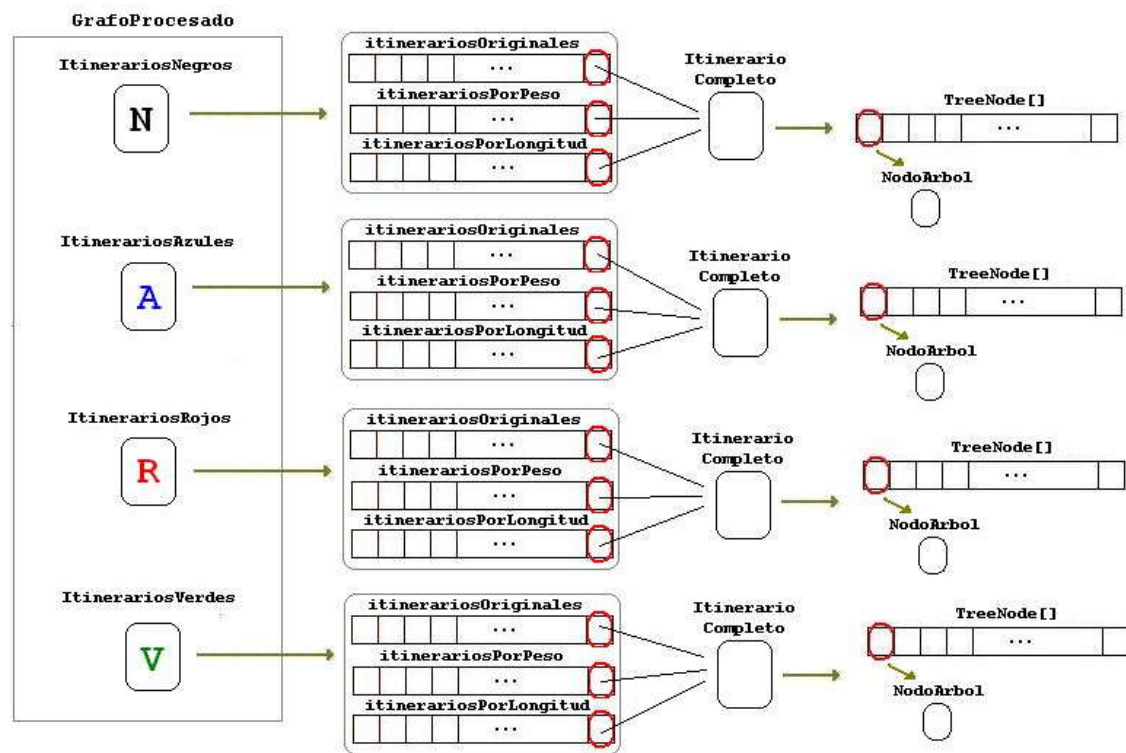
### **5.3.6 FASE VI: Procesado de datos para la generación del Documento Automático del Plan.**

La siguiente Fase de diseño consiste en procesar los datos obtenidos en las fases anteriores y procesarlos de forma que se pueda obtener un único objeto que sirva de parámetro a las clases destinadas a la generación del Documento Automático del Plan. Se trata de agrupar la lógica relativa a los distintos itinerarios y recursos del plan de forma que se puedan ir desarrollando las diferentes secciones del documento.

El objeto que recoge toda la información procesada se denomina **DatosProcesadosDocumentoODF** y sus atributos se recogen a continuación:

- **GrafoProcesado:** Objeto que recoge toda la información relacionada con los itinerarios del Plan, a partir de objetos específicos para cada grupo. Contiene, tanto los itinerarios originales, como los itinerarios ordenados por importancia y por longitud.
- **DatosPlan:** Objeto con los datos generales del Plan de Abonado:
  - Número de despliegue
  - Nombre
  - Versión
  - Fecha de creación
  - Autor
  - Descripción
  - Número de nodos
  - Número de recursos
- **RespuestasUsuario:** Objeto con las opciones elegidas por el usuario para generar el Documento Automático.
- **RecursosPlan:** Objeto con la información de los diferentes recursos del Plan.
- Vector general de nodos
- Ruta de la imagen del plan (opcional)

En la siguiente imagen se puede ver el procesado que sufren los datos hasta construir el objeto **GrafoProcesado**, que contiene la información principal relativa a los itinerarios.

Figura 5-21: Procesado de datos del objeto *GrafoProcesado*

### 5.3.7 FASE VII: Generar listados de referencia para todos los itinerarios

La séptima Fase de diseño consiste en la generación de listados de referencia con la información de los diferentes itinerarios del Plan de Abonado. Con la ejecución de la aplicación, se generará un fichero *.txt* con el listado de todos los itinerarios del plan. La única información que aparecerá en estos listados será una secuencia de nombres y tipos de nodo para cada uno de los itinerarios del plan. Dichos itinerarios estarán ordenados en función de la opción escogida por el usuario en la interfaz gráfica.

Los listados de referencia se generarán en la ruta */DocumentacionAutomatica/Configuracion* y su sintaxis será la siguiente:

- *<nombrePlan>\_listadoLon.txt*: Si se ha elegido ordenación por longitud.
- *<nombrePlan>\_listadoImp.txt*: Si se ha elegido ordenación por importancia.

La generación de listados de referencia permite al usuario visualizar de forma rápida la relación de nodos que compone cada uno de los itinerarios. De esta forma, un usuario puede escoger los itinerarios concretos que desea incluir en la documentación. Puede ocurrir, por ejemplo, que un usuario esté especialmente interesado en visualizar los itinerarios en los que existe un determinado Módulo de Lógica. En ese caso, mediante un vistazo rápido a los listados de referencia, puede localizar el número de itinerario en el que está dicho nodo e incluirlo en la documentación.

En la siguiente imagen se muestra el aspecto de un listado de referencia ordenado por longitud del plan *consultaHoroscopo.xml*.

**Figura 5-22: Listado de referencia ordenado por longitud**

```

***** ITINERARIOS CORRECTOS *****
ITINERARIO 1 (Longitud: 11)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
ConfirmacionZodiaco --> (nodo_locucion)
ServExterno_zodiaco --> (nodo_invocarServicioExterno)
Correcto --> (nodo_locucion)
Solucion --> (nodo_locucion)
algomas --> (nodo_datos_entrada)
if_1 --> (nodo_if)
Despedida --> (nodo_locucion)
Fin_1 --> (nodo_fin)

ITINERARIO 2 (Longitud: 10)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
ConfirmacionZodiaco --> (nodo_locucion)
ServExterno_zodiaco --> (nodo_invocarServicioExterno)
Correcto --> (nodo_locucion)
Solucion --> (nodo_locucion)
algomas --> (nodo_datos_entrada)
if_1 --> (nodo_if)
Bienvenido --> (nodo_locucion)

***** ITINERARIOS SIN SALIDA VALIDA *****
ITINERARIO 1 (Longitud: 11)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
ConfirmacionZodiaco --> (nodo_locucion)
ServExterno_zodiaco --> (nodo_invocarServicioExterno)
Correcto --> (nodo_locucion)
Solucion --> (nodo_locucion)
algomas --> (nodo_datos_entrada)
Max_intentos --> (nodo_locucion) --> (*)
Despedida --> (nodo_locucion)
Fin_1 --> (nodo_fin)

ITINERARIO 2 (Longitud: 6)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
Max_intentos --> (nodo_locucion) --> (*)
Despedida --> (nodo_locucion)
Fin_1 --> (nodo_fin)

***** ITINERARIOS CON TRANSICIONES DE ERROR *****
ITINERARIO 1 (Longitud: 10)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
ConfirmacionZodiaco --> (nodo_locucion)
ServExterno_zodiaco --> (nodo_invocarServicioExterno)
Correcto --> (nodo_locucion)
Solucion --> (nodo_locucion)
algomas --> (nodo_datos_entrada)
Fallo --> (nodo_locucion) --> (*)
algomas --> (nodo_datos_entrada)

ITINERARIO 2 (Longitud: 7)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
ConfirmacionZodiaco --> (nodo_locucion)
ServExterno_zodiaco --> (nodo_invocarServicioExterno)
Error --> (nodo_locucion) --> (*)
Bienvenido --> (nodo_locucion)

ITINERARIO 3 (Longitud: 5)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
Error --> (nodo_locucion) --> (*)
Bienvenido --> (nodo_locucion)

***** ITINERARIOS CON TRANSICIONES DE VUELTA ATRÁS *****
ITINERARIO 1 (Longitud: 10)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
ConfirmacionZodiaco --> (nodo_locucion)
ServExterno_zodiaco --> (nodo_invocarServicioExterno)
Correcto --> (nodo_locucion)
Solucion --> (nodo_locucion)
algomas --> (nodo_datos_entrada)
Fallo --> (nodo_locucion) --> (*)
algomas --> (nodo_datos_entrada)

ITINERARIO 2 (Longitud: 5)
NODO_INICIO --> (nodo_inicio)
Bienvenido --> (nodo_locucion)
EligeZodiaco --> (nodo_datos_entrada)
Error --> (nodo_locucion) --> (*)
Bienvenido --> (nodo_locucion)

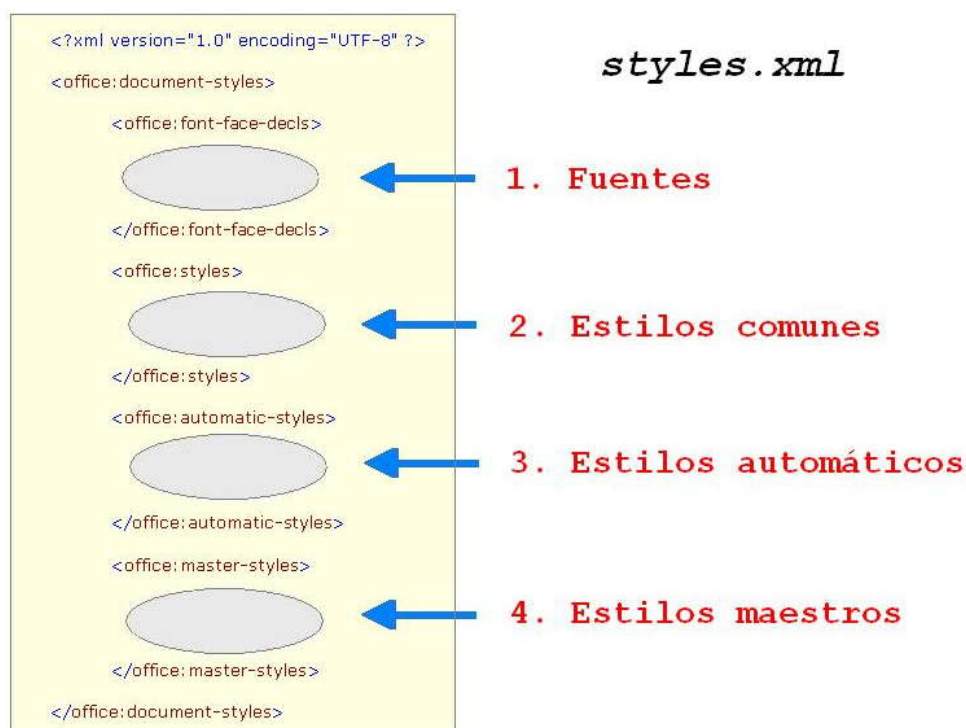
```

### 5.3.8 FASE VIII: Diseñar los estilos del Documento Automático del Plan

La siguiente Fase de diseño consiste en modelar los diferentes tipos de estilos que van a existir en el documento a generar. El método que gestiona el modelado de los estilos se denomina *cargarEstilos()* y pertenece a la clase **Plantilla**. En él se siguen los siguientes pasos para el tratamiento de los estilos:

- **Cargar encabezados y pies de página:** Configura el estilo y el texto de los encabezados y pies de página del documento. En el encabezado se sitúa una imagen con el logo de la empresa mientras que en el pie de página se coloca a la izquierda un título del documento y a la derecha el número de página. Los estilos para el encabezado y el pie de página se sitúan en el fichero *styles.xml*. Primero será necesario definir en los estilos automáticos del fichero, las “cajas” donde van a ir los encabezados y pies para luego, en segundo término, definir las propiedades concretas en los estilos maestros.
- **Cargar fuentes del documento:** Carga las distintas fuentes que se pueden utilizar en el documento. Es necesario introducir las nuevas fuentes tanto en el fichero *styles.xml* como en el fichero *content.xml*
- **Cambiar estilos por defecto:** Configura los estilos por defecto del documento. Dichos estilos actuarán cuando no se especifiquen otros en los diferentes elementos de contenido. Básicamente, los estilos por defecto que se ajustarán serán los relacionados con el estilo y el tamaño de la fuente.
- **Configurar estilos comunes:** Configura los estilos comunes del documento que se desea queden alojados en el fichero *styles.xml*. En la mayoría de los casos, suelen colocarse aquí los estilos padres a los que se referencia desde otros estilos automáticos del documento.

En la siguiente imagen, se puede observar donde se ha de introducir el código para el manejo de los estilos en el fichero *styles.xml*.

**Figura 5-23: Código de estilos en el fichero sytes.xml**

- **Configurar estilos automáticos:** Crea la mayor parte de los estilos. Se añaden al fichero *content.xml*, de forma que su manipulación resulta mucho más rápida. Se agrupan en familias dependiendo del ámbito sobre el que aplican:
  - Estilos de **PÁRRAFO**: Tienen como ámbito un párrafo de texto. Un ejemplo puede ser el tipo de alineación: izquierda, centrada, derecha o justificada.
  - Estilos de **TEXTO**: Tienen como ámbito un fragmento de texto. Sirven por ejemplo, para resaltar un texto en negrita, cursiva o subrayado.
  - Estilos de **TABLA**: Afectan a las tablas del documento. Establecen parámetros como pueden ser la anchura de la tabla o el espacio con el texto que la precede.

- Estilos de **FILA**: Afecta a las filas de una tabla. Permite establecer características comunes referentes al ámbito de fila de una tabla.
- Estilos de **CELDA**: Aplican a una celda concreta de una tabla. Permite dotar una celda de características concretas, como puede ser el color de fondo o el *padding* del texto que hay contenido en ella.
- Estilos de **SECCIÓN**: Tienen como ámbito una sección concreta del documento.
- Estilos de **GRÁFICO**: Tienen como ámbito el marco en el que se inscribe una determinada imagen en el documento. Permite modificar, por ejemplo, la altura y la anchura del frame que alberga una determinada imagen.

En la siguiente tabla se recogen los principales estilos que se han utilizado para elaborar el Documento Automático del Plan de Abonado.

**Tabla 5-4: Tabla de estilos del Documento Automático del Plan**

TABLA DE ESTILOS DEL DOCUMENTO		
Familia	Nombre	Descripción
	PORTADA	Marcos que contiene la portada del documento
	TITULO_MANUAL	Estilo del título del documento
	TITULO_1	Títulos para encabezados de primer nivel
	TITULO_2	Títulos para encabezados de segundo nivel
	TITULO_3	Títulos para encabezados de tercer nivel
	ALIN_IZDA	Alineación a la izquierda



<b>PÁRRAFO</b>	ALIN_CENTRO	Alineación centrada
	ALIN_DCHA	Alineación a la derecha
	ALIN_JUSTIFICADA	Alineación justificada
	SALTAR_PAGINA	Estilos para salto de página
	PARA_INDICE	Estilos para entradas de índice
	PADRE_PORTADA	Estilo padre de la portada
	STANDARD	Estilo estándar a aplicar ante la ausencia de otros estilos
	PAG_MAESTRA_PORTADA	Estilos de la página maestra de la portada del documento
	ENCABEZADO_PAG_PORTADA	Encabezado de la portada
	PIE_PAG_PORTADA	Pie de la portada
	PIE_PAG_ESTANDAR	Pie estándar
	TITULO_INDICE	Estilo del título del índice
	SCRIPT	Estilo para bloques de código
<b>TEXTO</b>	NEGRITA	Fuente negrita
	CURSIVA	Fuente cursiva
	SUBRAYADO	Fuente subrayado
	FUENTE_AZUL	Color azul de fuente
	FUENTE_AZUL_CURSIVA	Color azul y estilo cursiva
	FUENTE_ROJA	Color roja de fuente
	FUENTE_ROJA_CURSIVA	Color roja y cursiva
	FUENTE_VERDE	Color verde de fuente
	FUENTE_VERDE_CURSIVA	Color verde y cursiva
	TABLA_ITINERARIO	Estilos para la tabla de un itinerario del Plan

<b>TABLA</b>	TABLA_BASE_DATOS	Estilos para las tablas de los recursos de tipo Base de Datos
	TABLA_GRAMATICA	Estilos para las tablas de los recursos de tipo Gramática
	TABLA_SERV_EXTERNO	Estilos para las tablas de los recursos de tipo Serv. Externo
<b>FILA</b>	FILA_POR_DEFEECTO	Estilo de fila por defecto
<b>CELDA</b>	FILA_ENCABEZADO	Estilo para celdas combinadas
	CELDA_POR_DEFEECTO	Estilo de celda por defecto
<b>SECCIÓN</b>	SECT1	Estilos para secciones de nivel uno
<b>GRÁFICO</b>	FRAME_IMAGEN_PORTADA	Estilos para el marco de la imagen de la portada
	FRAME_TITULO_PORTADA	Estilos para el marco del título de la portada
	PADRE_FRAME_IMAGEN_PORTADA	Estilo padre de la imagen
	PADRE_FRAME_TITULO_PORTADA	Estilo padre del título

### 5.3.9 FASE IX: Diseñar el contenido del Documento Automático del Plan

Una vez establecidos los estilos del Documento en la Fase de diseño anterior, es necesario abordar como última Fase de diseño el contenido del Documento Automático del Plan. El método que se encarga de ello se denomina **rellenarDocumentoODF()** y se encuentra en la clase **Plantilla**. Dicho método cubre los siguientes pasos:

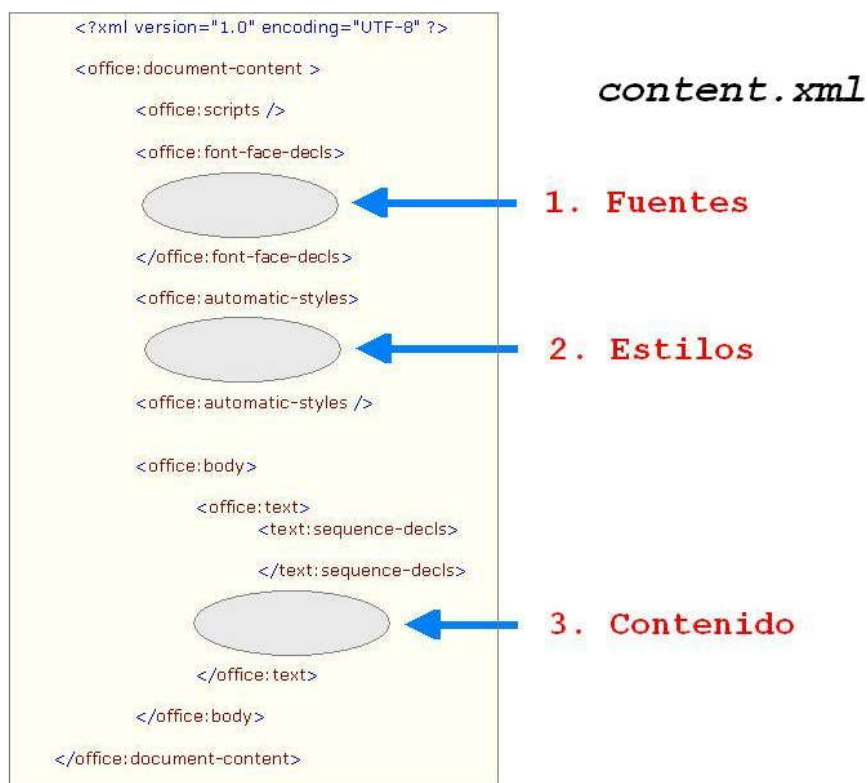
- **Añadir imágenes del documento:** Lógica que permite añadir al paquete de ficheros del documento *.odt* las distintas imágenes que se van a utilizar a lo largo del documento. Todas las imágenes se almacenan en el directorio */Pictures*.
- **Crear portada del documento:** Lógica que permite crear la portada del documento. Estará compuesta por dos frames, uno a la izquierda con el

logotipo de la empresa y otro centrado a la derecha, con el título del documento. En el título del documento se incluirá el nombre y la versión del Plan de Abonado.

- **Crear sección de Datos Generales del Plan:** Lógica que permite crear una sección donde se puede visualizar la información contenida en el objeto **DatosPlan**. Tendrá carácter obligatorio y será siempre la primera sección del documento tras el índice de contenidos. Contendrá una lista de ítems con los datos principales del Plan de Abonado.
- **Crear sección de Grafo Completo del Plan:** Lógica que permite crear una sección donde se puede ver el esquema del grafo del Plan. Tendrá carácter opcional y estará supeditada a la introducción por parte del usuario de un fichero con la imagen del Plan de Abonado. La sección contendrá un párrafo explicativo y la imagen del grafo del Plan, que previamente ha debido ser añadida en el primer paso de esta fase de diseño.
- **Crear secciones de Itinerarios:** Lógica que permite crear una sección por cada grupo de itinerarios. Todas las secciones de este tipo tienen carácter opcional y dependen de las opciones elegidas por el usuario en la interfaz gráfica. Si el usuario ha elegido como opción **“Ver itinerario de mayor importancia / longitud”**, se creará en primer lugar una sección con el itinerario principal. Todas las secciones contendrán un párrafo explicativo de primer nivel. A continuación habrá una subsección por cada itinerario que exista. Dentro de la subsección se colocará una tabla con el listado de todos los nodos del itinerario y a continuación habrá una sección de tercer nivel con la información a nivel de nodo.
- **Crear sección de Recursos:** Lógica que permite crear la sección de Recursos del Plan. Tendrá carácter obligatorio y contendrá una subsección por cada grupo de recursos distintos que existan.

- **Crear anexo:** Lógica que permite crear una sección de “Anexo en el Documento del Plan”. Se utiliza para crear un apartado en el que consten las opciones elegidas por el usuario para generar la documentación automática.
- **Crear índice:** Lógica que permite crear el Índice de Contenidos del Documento. Se crea en el último paso. Recorremos el árbol *DOM* del fichero *content.xml* hasta situar la sección del índice entre la portada y la primera sección del documento.
- **Gestionar índices de secciones:** Lógica que permite gestionar el índice general de secciones. Se modela mediante una propiedad estática en la clase *Plantilla*. Permite mantener de forma coherente el número de cada una de las secciones del documento, teniendo en cuenta que varía de forma dinámica en función de los deseos del usuario.
- **Salto de página:** Lógica que permite introducir saltos de página manuales en el documento, de forma que se puedan separar cada una de las diferentes secciones.

En la siguiente imagen, se muestra donde se ha de introducir el código para rellenar el documento *.odt*, en el fichero *content.xml*:

**Figura 5-24: Código del contenido en el fichero content.xml**



---

## 6 IMPLEMENTACIÓN Y PRUEBAS

### 6.1 INTRODUCCIÓN

En el siguiente capítulo se va a hacer un estudio de los aspectos más importantes de la implementación del sistema. En primer lugar, se describirá cómo se ha codificado el sistema. En segundo lugar se verá cómo se han implementado los bloques más simples de lógica relacionados con el Documento ODF. Se analizará entre otras cosas cómo se añade una imagen, cómo se inserta un párrafo, una lista, una tabla, etc. A continuación se verá la estructura del Documento Automático, con sus diferentes secciones, aplicado al Plan de Abonado *consultarHoroscopo.xml*. Más tarde se analizará detenidamente el paquete de la aplicación así como su sistema de trazas, control de excepciones y pruebas de sistema.

### 6.2 CODIFICACIÓN DEL SISTEMA

La presente herramienta de “*Generación Automática de Documentación de Planes de Abonado*” ha sido codificada utilizando como lenguaje de programación **JAVA**. Se ha utilizado la versión 1.5 de JDK. Los aspectos más significativos que tienen que ver con la codificación se describen a continuación:

- El mapeo del fichero de Plan de Abonado (*.xml*) a objetos Java se ha realizado utilizando el parser XML denominado “*Xerces Java Parser*”, contenido en la librería **XercesImpl.jar**.
- La estructura de datos que modela cada uno de los Planes de Abonado en forma de árbol, se ha codificado con objetos de la clase **JTree**, de forma que cada uno de los itinerarios contenidos en los vectores de grupo está formado por objetos de la clase **TreeNode[]**. En última instancia, cada uno de los Módulos de Lógica que forman los itinerarios se han codificado utilizando la

clase **DefaultMutableTreeNode** cuyo valor contenido es una instancia de la clase **NodoArbol**.

- El diseño de la interfaz gráfica se ha llevado a cabo utilizando **JSWING**, mediante el uso de la librería **swing-layout-1\_0\_3.jar**.
- El acceso, la creación y la manipulación de ficheros ODF desde código Java se ha llevado a cabo a partir del **API ODFDOM**.

### **6.2.1 Documentos personalizados. Interfaz Gráfica de Usuario**

La presente herramienta de documentación dispone de una interfaz gráfica de usuario que permite a los clientes seleccionar las opciones que deseen a la hora de confeccionar el Documento Automático de su Plan de Abonado. Las opciones que se pueden elegir se detallan a continuación:



**Figura 6-1: Interfaz Gráfica. Documentos personalizados**

Sistema Documentacion FACIL - Interfaz gráfica de usuario

Número de itinerarios correctos: 2

Número de itinerarios sin salida válida: 2

Número de itinerarios con errores: 3

Número de itinerarios con eventos de marcha atrás: 2

---

Número de despliegue:  Ver todos los itinerarios del plan ☐

Tipo de ordenacion

Importancia ☐ Longitud ☐

Ver el itinerario de mayor importancia / longitud ☐

Ver todos los itinerarios correctos del plan ☐ Ver sólo algunos ...

Ver todos los itinerarios sin salida válida del plan ☐ Ver sólo algunos ...

Ver todos los itinerarios con errores del plan ☐ Ver sólo algunos ...

Ver todos los itinerarios con eventos de marcha atrás en el plan ☐ Ver sólo algunos ...

Aceptar

- **Número de itinerarios de cada grupo:** En la zona superior de la interfaz, está disponible la información del número de itinerarios que existen de cada tipo en el Plan de Abonado.
- **Número de despliegue:** Se debe rellenar obligatoriamente. Indica de modo informativo el número 90X en el que se va a desplegar el Servicio.
- **Tipo de ordenación:** Tiene carácter obligatorio. Se pueden ordenar los itinerarios por longitud o por importancia.
- **Ver todos los itinerarios del plan.** Crea el documento automático más completo posible con la información de todos y cada uno de los itinerarios que existen en el Plan de Abonado. Su selección deshabilita el resto de opciones que se encuentran más abajo.

- **Ver el itinerario de mayor importancia / longitud:** Crea en el documento automático una sección destinada al itinerario principal del Plan de Abonado según el criterio de ordenación elegido más arriba.
- **Ver todos lo itinerarios correctos del plan:** Permite crear la sección con todos los itinerarios correctos del Plan de Abonado.
- **Ver todos los itinerarios sin salida válida del plan:** Permite crear la sección con todos los itinerarios que tengan alguna transición azul en el Plan.
- **Ver todos los itinerarios con errores del plan:** Permite crear la sección con todos los itinerarios con eventos de error en el Plan.
- **Ver todos los itinerarios con eventos de marcha atrás en el plan:** Permite crear la sección con todos los itinerarios que tengan alguna transición verde en el Plan.
- **Ver sólo algunos ...:** Permite establecer qué itinerarios concretos deseamos incluir en la documentación. La relación de itinerarios a incluir se dispondrá colocando los números de los itinerarios separados por comas. Esta opción permanecerá deshabilitada si se ha elegido ver todos los itinerarios de ese grupo.

Se puede consultar más información relacionada con la Interfaz Gráfica de Usuario en el Manual de Usuario descrito en el **ANEXO B MANUAL DE USUARIO**.

## **6.3 IMPLEMENTACIÓN DEL DOCUMENTO ODF**

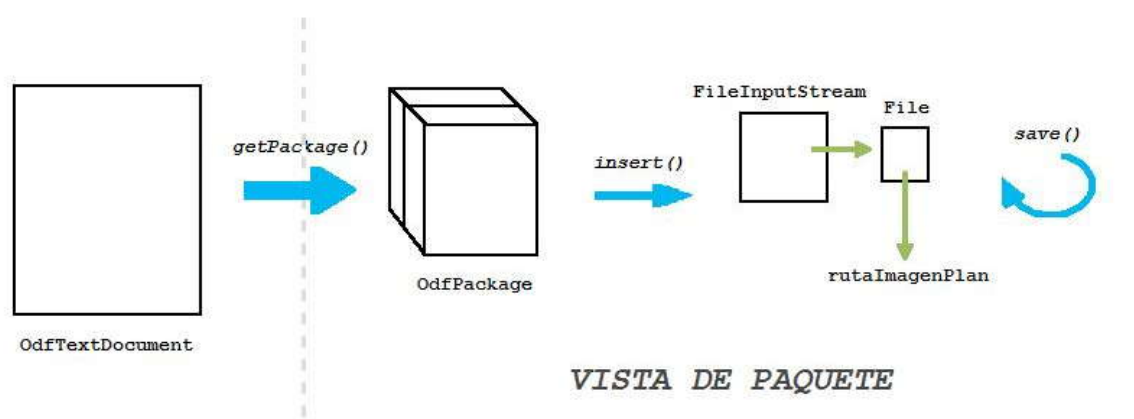
### **6.3.1 Añadir una imagen al paquete .odt**

Para añadir una imagen al paquete del fichero *.odt*, es necesario utilizar las clases y métodos de la primera capa del API ODFDOM, tal y como se explica en el apartado **3.3.2.3 Estándar de documentación abierto OpenDocument (ODF)**.

Una vez creado el documento de texto en blanco, se puede acceder al paquete del documento a través del método *getPackage()* de la clase **OdfPackage**. La primera capa del modelo ODFDOM permite manipular los ficheros comprimidos en la estructura de directorios que conforman el paquete. Las imágenes se incluirán todas en el directorio **/Pictures** y será imprescindible añadirlas a la estructura de ficheros para poder utilizarlas luego en cualquier sección del documento.

En la siguiente figura se muestra el mecanismo para añadir imágenes al paquete *.odt*.

**Figura 6-2: Añadir imágenes al paquete *.odt***



### 6.3.2 Añadir elementos al documento *.odt*

Para añadir elementos al documento *.odt*, se va a hacer uso de los objetos existentes en la tercera capa del modelo ODFDOM. Básicamente, las acciones a seguir para insertar cualquier elemento en el documento *.odt* de la aplicación son las siguientes:

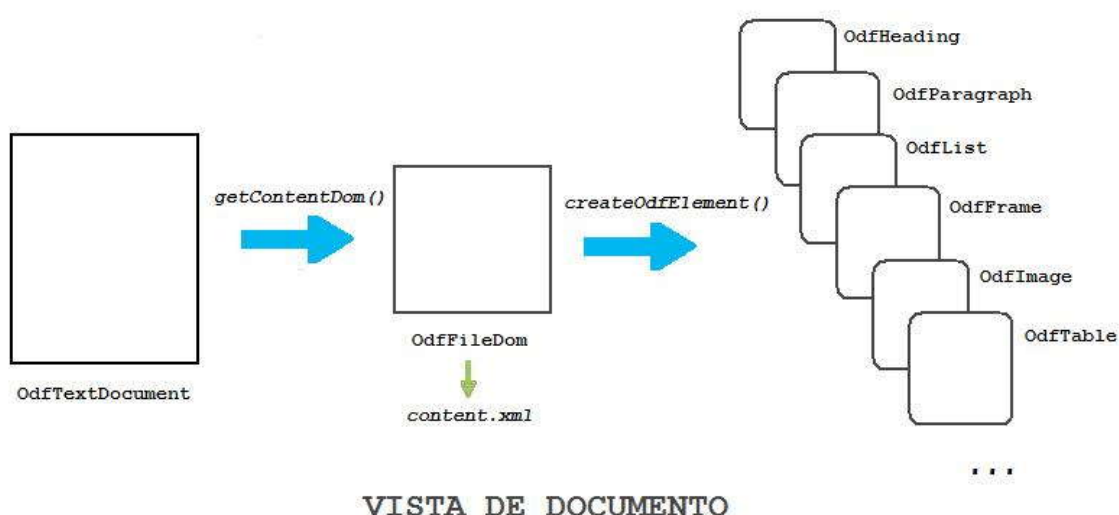
- Se crea el documento de texto en blanco (objeto **OdfTextDocument**) y se le añaden las imágenes necesarias como se ha descrito en el apartado anterior.
- A partir del método *getContentDom()* de la clase **OdfTextDocument**, accedemos al árbol DOM del fichero *content.xml* (objeto **OdfFileDom**), donde habrá que ir insertando los bloques de código que se vayan generando. Los elementos se incorporarán al árbol a través del método *appendChild()*,

al igual que ocurre con cualquier otro árbol DOM. Para movernos por el árbol del documento se utilizará la sintaxis descrita por *XPath*.

- El método estático *createOdfElement()* de la clase **OdfFileDom**, permite crear cualquier tipo de elemento que le indiquemos por parámetro. Todos los elementos tienen un objeto representativo en el API y una traducción automática y transparente a código xml que se insertará en el fichero de contenido del paquete.
- Una vez creado el elemento, típicamente se le podrá asignar un determinado estilo. Un elemento puede contener, a su vez, otros elementos en una jerarquía inferior, por lo que hay que poner especial cuidado en la forma en la que se incorporan los objetos al árbol del documento. La especificación completa de OpenDocument v1.1 se puede consultar en el manual *OpenDocument\_v1\_1\_ES.pdf* cuyo enlace se incluye en la bibliografía [1 **Formato de Documento Abierto para Aplicaciones Ofimáticas**].

En la siguiente imagen se representan todos estos pasos.

**Figura 6-3: Añadir elementos al documento .odt**



### 6.3.2.1 Insertar estilo. Objeto **OdfStyle**

El objeto **OdfStyle** permite insertar un elemento de estilo en el documento. Como hemos visto en apartados anteriores, los elementos de estilo pueden ubicarse tanto en el fichero *styles.xml* como en el fichero *content.xml*. Dependiendo de la ubicación que tengan los estilos y del tipo de estilo que se desee generar se utilizarán objetos distintos, extendiendo todos ellos de la clase **OdfStyle**.

El objeto **OdfStyle** presenta las siguientes propiedades:

- **displayName**: Nombre que presentará el estilo en el display de la herramienta ofimática.
- **styleName**: Nombre del estilo
- **styleFamily**: Nombre de la familia a la que pertenece el estilo.

Una vez definidas los atributos del estilo, es necesario indicar sus características concretas. Para ello, será necesaria la implementación de un objeto de propiedades que será diferente en función del tipo de propiedades que se desee configurar. A continuación se describen los objetos más importantes:

- **OdfParagraphProperties**: Objeto que permite definir las propiedades de estilo relativas a la familia Paragraph, como pueden ser los márgenes, el padding, mantener todas las líneas juntas, tipo de alineación, etc.
- **OdfTextProperties**: Objeto que permite definir las propiedades de estilo relativas a la familia Text, como pueden ser el tipo y tamaño de la fuente, color, etc.
- **OdfTableProperties**: Objeto que define propiedades de estilo relativas a la familia Table, como puede ser la anchura, color de fondo o alineación del texto.

En el siguiente ejemplo, se puede ver el código xml del estilo asociado al Título del Documento Automático.

```
<style:style style:display-name="Titulo_manual"
style:family="paragraph" style:name="Titulo_manual">

    <style:paragraph-properties fo:orphans="2"
fo:widows="2" />

    <style:text-properties fo:color="#00086e"
fo:country="ES" fo:font-size="26pt" fo:font-
weight="bold" fo:language="es" style:country-
complex="SA" style:font-name="Verdana" style:font-
name-asian="Times New Roman" style:font-name-
complex="Times New Roman" style:font-size-
asian="26pt" style:font-size-complex="40pt"
style:font-weight-asian="bold" style:language-
complex="ar" style:letter-kerning="true" />

</style:style>
```

#### 6.3.2.2 Insertar cabecera. Objeto OdfHeading

El objeto **OdfHeading** permite insertar un encabezado en el documento. Sus parámetros y métodos más importantes son:

- ***isListHeader***: Atributo booleano, cuyo valor true indica que es visible la numeración del encabezado, manteniéndose escondida en caso contrario.
- ***outlineLevel***: Atributo de tipo Integer. Indica el nivel del encabezado. El primer nivel será 1, el segundo será 2 y así sucesivamente.
- ***setStyleName(String estilo)***: Método con el que se le puede asignar un estilo al encabezado.

Un ejemplo del código xml generado para un encabezado de segundo nivel, puede ser el siguiente:

```
<text:h text:is-list-header="false" text:outline-level="2"
text:style-name="Titulo_2">
```

### 3.1. ITINERARIO CORRECTO NÚMERO 1

```
</text:h>
```

#### 6.3.2.3 Insertar párrafo. Objeto OdfParagraph

El objeto **OdfParagraph** permite insertar párrafos de texto en el documento. Posee un método para dotar a los párrafos de un determinado estilo. El texto que contiene un párrafo está contenido en un nodo hijo del objeto OdfParagraph, por lo que es necesario crear previamente un nodo de texto, enlazarlo con el objeto OdfParagraph y posteriormente hacer colgar el objeto del párrafo en la posición del árbol DOM que se desee. Para gestionar este tipo de nodos tenemos dos métodos dentro de la aplicación:

- *text(OdfElement parent, String text)*: Crea un nodo con el texto del segundo parámetro y lo enlaza con el elemento padre. OdfParagraph extiende de OdfElement. Un ejemplo de código xml generado a partir de este método es el siguiente:

```
<text:p text:style-name="AlinearCentro">
```

ESQUEMA DEL ITINERARIO

```
</text:p>
```

- *text(OdfElement parent, String text, String estilo)*: Crea un nodo con el texto del segundo parámetro aplicándole el estilo descrito en el tercer parámetro. Para aplicar el estilo al texto del párrafo, es necesario envolver el nodo de texto en un elemento **OdfSpan**, de modo parecido a lo que ocurre en HTML. Con este método podemos conseguir, por ejemplo, resaltar en negrita una palabra dentro del párrafo.

```

<text:p>
    Nombre :
    <text:span text:style-name="Negrita">
        ConsultaHoroscopo
    </text:span>
</text:p>

```

#### 6.3.2.4 Insertar salto de página. Objeto OdfPageBreak

Para introducir un salto de página en el documento, se hace uso del objeto **OdfPageBreak**. La característica del salto de página está asociada al estilo que se asigna al párrafo. Al igual que en los casos anteriores el estilo se asigna mediante el método *setStyleName(String estilo)*. A continuación se muestra el código xml, tanto del elemento ODF como del elemento de estilo caracterizado por un objeto de tipo **OdfStyleCustom**.

```

<text:p text:style-name="SaltarPagina" />

<style:style style:family="paragraph"
style:name="SaltarPagina">
    <style:paragraph-properties fo:break-before="page"
/>
</style:style>

```

El elemento de estilo se inserta dentro de los estilos automáticos del fichero *content.xml*.

#### 6.3.2.5 Insertar lista. Objeto OdfList

Para introducir una lista en el documento, es necesario implementar el objeto **OdfList**. Cada uno de los elementos de la lista se modela a partir de un objeto de tipo **OdfListItem**, de forma que cada uno de estos



objetos cuelga jerárquicamente del nodo raíz de la lista. Típicamente, se utilizarán párrafos para el contenido de cada ítem.

El siguiente código ilustra un ejemplo de lista con algunos de los datos generales del Plan de Abonado.

```
<text:list>
  <text:list-item>
    <text:p>
      Número 90X de despliegue:
      <text:span text:style-
name="Negrita">900202020</text:span>
    </text:p>
  </text:list-item>
  <text:list-item>
    <text:p>
      Nombre:
      <text:span text:style-
name="Negrita">ConsultaHoroscopo</text:span>
    </text:p>
  </text:list-item>
  <text:list-item>
    <text:p>
      Versión:
      <text:span text:style-
name="Negrita">1.0</text:span>
    </text:p>
  </text:list-item>
  <text:list-item>
    <text:p>
      Fecha de creación:
      <text:span text:style-
name="Negrita">2009-04-07 09:59:18</text:span>
    </text:p>
  </text:list-item>
```

```

<text:list-item>
    <text:p>
        Autor:
        <text:span text:style-
name="Negrita">Anonimo</text:span>
    </text:p>
</text:list-item>
</text:list>

```

### 6.3.2.6 Insertar tabla. Objeto OdfTable

Para incluir una tabla en el documento, hay que hacer uso del objeto **OdfTable**. Dicho objeto permite modelar la estructura general de la tabla y dotarla de un determinado estilo. Cada una de sus filas, columnas y celdas se modela con un objeto diferente:

- **OdfTableColumn**: Define las características y propiedades relativas a las columnas de la tabla.
- **OdfTableRow**: Define las propiedades relacionadas con las filas de la tabla.
- **OdfTableCell**: Define las propiedades de cada una de las celdas de la tabla

En el siguiente ejemplo se presenta el código xml de una tabla de dos filas y dos columnas, donde las celdas de la primera fila están combinadas para formar el título de la tabla. Dicha tabla corresponde al objeto **PETICIÓN** del recurso de **Servicio Externo** que posee el plan *consultaHoroscopo.axml*.

```

<table:table table:style-
name="TablaPeticionServicioExterno">
    <table:table-column />
    <table:table-column />

```

```

<table:table-row>

  <table:table-cell table:number-columns-
spanned="2" table:style-name="FilaEncabezado">

    <text:p text:style-
name="AlinearCentro">CAMPOS DE LA PETICIÓN:
cz</text:p>

  </table:table-cell>

  <table:covered-table-cell />

</table:table-row>

<table:table-row table:style-name="FilaPorDefecto">

  <table:table-cell table:style-
name="CeldaPorDefecto">

    <text:p>

      <text:span text:style-
name="Negrita">ZODIACO</text:span>

    </text:p>

  </table:table-cell>

  <table:table-cell table:style-
name="CeldaPorDefecto">

    <text:p>CADENA</text:p>

  </table:table-cell>

</table:table-row>

</table:table>

```

#### 6.3.2.7 Insertar una imagen. Objeto OdfFrame + OdfImage

Para insertar una imagen en el documento se utilizan conjuntamente los objetos **OdfFrame** y **OdfImage**. Básicamente, el segundo objeto tiene una jerarquía inferior y requiere estar contenido dentro de un “frame”, para que funcione correctamente. Con el objeto OdfFrame se definen las propiedades del “marco” que contendrá la imagen, mientras que el objeto OdfImage es el que define la imagen en sí.

El método más importante del objeto `OdflImage` es *`setHref(String link)`*, método que permite establecer que imagen del directorio **/Pictures** se va a cargar.

En el siguiente ejemplo se muestra el código xml en el que se inserta la imagen del Plan de Abonado. Notar que para que la imagen se inserte correctamente es necesario que el objeto `OdffFrame` esté contenido en la jerarquía de un objeto de párrafo.

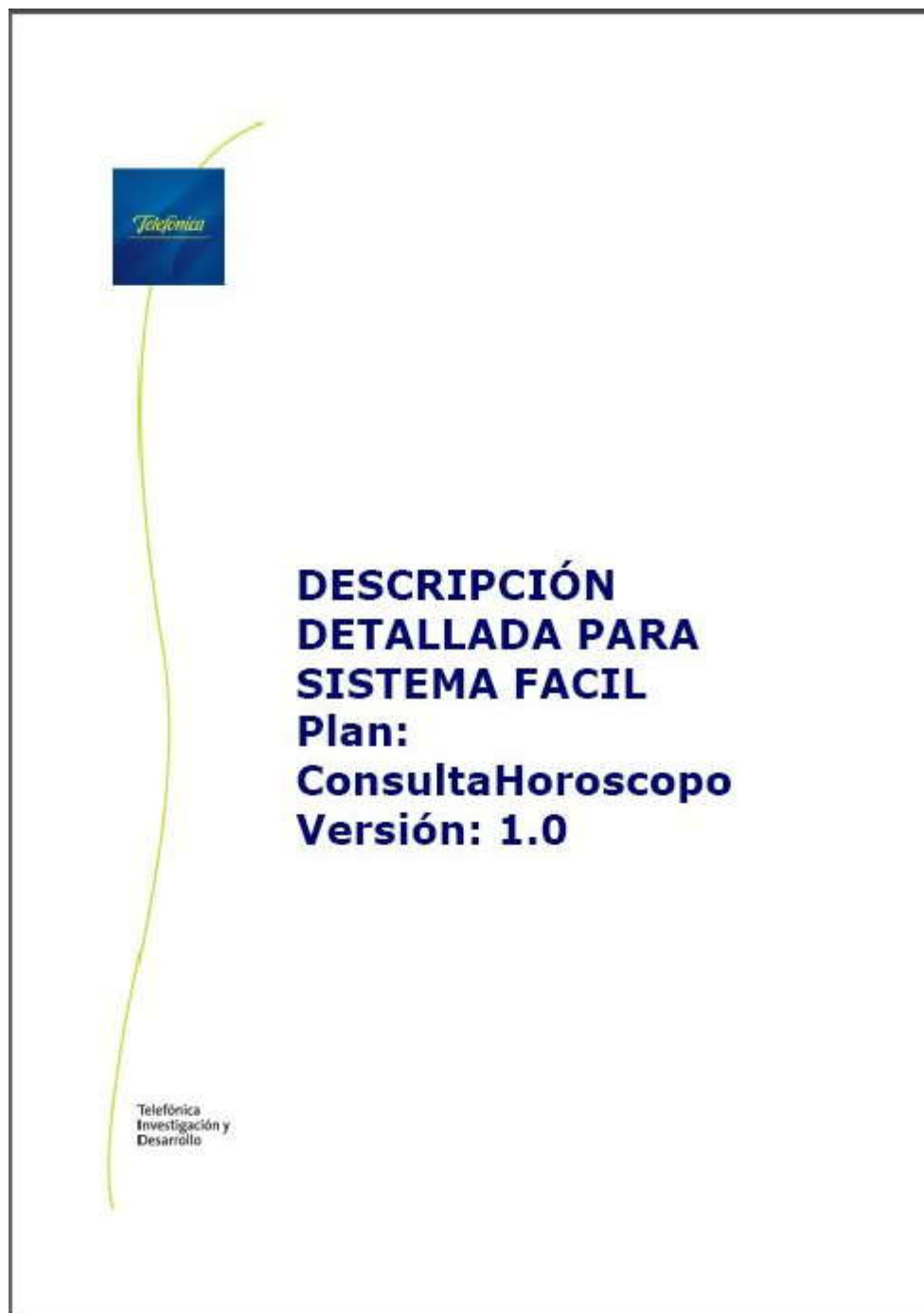
```
<text:p>
    <draw:frame svg:height="12.44cm" svg:width="15cm"
      text:anchor-type="paragraph">
        <draw:image xlink:actuate="onLoad"
          xlin:href="Pictures/ConsultaHoroscopo.png"
          xlink:show="embed" xlink:type="simple" />
      </draw:frame>
    </text:p>
```

## 6.4 ESTRUCTURA DEL DOCUMENTO AUTOMÁTICO

### 6.4.1 Portada del documento

La portada del documento tiene carácter obligatorio. Presentará en el margen izquierdo el logotipo de la empresa y en el centro, junto con el título de la herramienta, se colocará el nombre y la versión del Plan de Abonado para el que está generado. El Sistema AFABLE, por motivos de marketing, se denomina de cara al cliente con el nombre de Sistema FACIL, de ahí que aparezca con este nombre en todas las referencias del Documento Automático.

**Figura 6-4: Portada del Documento Automático**





### 6.4.2 Índice del documento

Tiene carácter obligatorio. Indicará el nombre de las distintas secciones y la página del documento en la que encontrarlas. Para construir el índice es necesario insertar marcas específicas al comienzo de cada sección de modo que puedan ser reconocidas al crear un objeto de tipo “*Tabla de Contenidos*”, que es la manera natural en la que se construye un índice en OpenDocument.

Las posibles entradas para el índice son las siguientes:

- Datos del plan (obligatorio)
- Grafo completo del plan (opcional)
- Análisis del itinerario principal del plan (opcional)
- Análisis de los itinerarios correctos del plan (opcional)
- Análisis de los itinerarios sin salida válida del plan (opcional)
- Análisis de los itinerarios con errores del plan (opcional)
- Análisis de los itinerarios con eventos de marcha atrás en el plan (opcional)
- Estudio de los recursos del plan (obligatorio)
- ANEXO: Interfaz de usuario (obligatorio)

**Figura 6-5: Índice del documento**

---

## Índice de contenidos

1. DATOS DEL PLAN.....	3
2. GRAFO COMPLETO DEL PLAN.....	4
3. ANÁLISIS DE LOS ITINERARIOS CORRECTOS DEL PLAN.....	5
4. ANÁLISIS DE LOS ITINERARIOS SIN SALIDA VÁLIDA DEL PLAN.....	14
5. ANÁLISIS DE LOS ITINERARIOS CON ERRORES DEL PLAN.....	21
6. ANÁLISIS DE LOS ITINERARIOS CON EVENTOS DE MARCHA ATRÁS EN EL PLAN. .	29
7. ESTUDIO DE LOS RECURSOS DEL PLAN.....	35
A.-) ANEXO: INTERFAZ DE USUARIO.....	41

---

DOCUMENTACIÓN AUTOMÁTICA GENERADA PARA EL SISTEMA FACIL 2 de 41

### 6.4.3 Datos del Plan

Esta sección tiene también carácter obligatorio. Será siempre la primera sección del documento y en ella podremos encontrar aspectos generales del Plan de Abonado. Dichos aspectos sirven al usuario para visualizar de forma resumida las siguientes cuestiones:

- **Número 90X de despliegue:** Recogido directamente de la interfaz de usuario. Tiene carácter informativo. Es el número en el que se despliega el Plan de Abonado.
- **Nombre:** Nombre del Plan de Abonado
- **Versión:** Versión con la que se ha creado el Plan de Abonado desde el AFABLE\_IDE.
- **Fecha de creación:** Fecha en la que se guardó por primera vez el Plan de Abonado.
- **Autor:** Autor de la persona o entidad responsable del diseño del Plan de Abonado en el AFABLE\_IDE.
- **Descripción** (opcional): Descripción del Plan de Abonado. Recoge un resumen de lo que intenta cubrir el Plan.
- **Número de nodos:** Número total de nodos del Plan de Abonado
- **Número de recursos:** Número total de recursos empleados en el Plan de Abonado.
- **Número de itinerarios totales:** Número total de caminos por los que puede cursarse una llamada sobre el Plan de Abonado
- **Número de itinerarios correctos:** Número de posibles caminos en los que la llamada evoluciona con transiciones correctas entre los distintos nodos. Todos los itinerarios correctos cumplen una misión “*deseada*” para la llamada por el diseñador del Plan.



- **Número de itinerarios sin salida válida:** Número de posibles caminos en los que la llamada sufre en alguno de sus nodos un evento de tipo “no-input” ò “no-match” por lo que evoluciona a un nodo diferente formando itinerarios “*no deseados*” pero “*esperables*” por el diseñador del Plan.
- **Número de itinerarios con errores:** Número de posibles caminos en los que la llamada sufre en alguno de sus nodos un evento de error “*no esperado*” por el diseñador del Plan. Dichos eventos provocan una transición hacia nodos que se encargan de avisar al usuario del error.
- **Número de itinerarios con eventos de marcha atrás:** Número de posibles caminos en los que existe al menos un nodo en el que el usuario puede voluntariamente volver hacia atrás, haciendo que la llamada curse un nodo anterior al actual.

En la siguiente figura se recoge el aspecto que tiene esta sección en el documento del Plan de Abonado.

**Figura 6-6: Datos del Plan de Abonado**

*Telefónica*

Telefónica  
Investigación y  
Desarrollo

### 1. DATOS DEL PLAN

- Número 90X de despliegue: 900202020
- Nombre: ConsultaHoroscopo
- Versión: 1.0
- Fecha de creación: 2009-04-07 09:59:18
- Autor: Anonimo
- Descripción: "Consulta el signo del zodiaco"
- Número de nodos: 14
- Número de recursos: 3
- Número de itinerarios totales: 9
- Número de itinerarios correctos: 2
- Número de itinerarios sin salida válida: 2
- Número de itinerarios con errores: 3
- Número de itinerarios con eventos de marcha atrás: 2

---

DOCUMENTACIÓN AUTOMÁTICA GENERADA PARA EL SISTEMA FACIL

3 de 41



### 6.4.5 Itinerario principal del Plan

Esta sección es opcional. Por itinerario principal del Plan de Abonado se entiende el primero de los itinerarios correctos, ordenados por importancia o longitud según escoja el usuario en la interfaz gráfica al comienzo de la aplicación. La sección consta de un primer párrafo explicativo, seguido de una tabla con los nodos que conforman el itinerario. A continuación de la tabla se explica detalladamente la información más relevante de cada nodo. En la siguiente figura, se muestra el ejemplo de itinerario principal para el Plan *consultaHoroscopo.xml*.

**Figura 6-8: Tabla de nodos del itinerario principal**

2. ITINERARIO PRINCIPAL DEL PLAN	
En la figura que se muestra a continuación aparece el esquema del itinerario principal del plan. El itinerario principal es el primero de la lista de itinerarios correctos.	
ESQUEMA DEL ITINERARIO	
	NODO_INICIO
	Bienvenido
	EligeZodiaco
	ConfirmacionZodiaco
	ServExterno_zodiaco
	Correcto
	Solucion
	algomas
	if_I
	Despedida
	Fm_I

### 6.4.6 Análisis de los itinerarios correctos del Plan

Esta sección es opcional y su inclusión dependerá de las opciones elegidas por el usuario en la interfaz gráfica presentada al principio de la aplicación. En primer

lugar, se puede observar un párrafo común explicativo de este tipo de itinerarios. En segundo lugar se plantearán por cada itinerario existente, una subsección en la que se tratará cada uno de los nodos que conforman el itinerario con la información específica de cada tipo de nodo. Si el usuario ha optado por ver un subconjunto determinado de itinerarios se presentarán sólo estos en la documentación.

A modo de ejemplo, en la siguiente figura se recoge la información de un nodo de tipo **Locucion**. Para este tipo de nodos, la información relevante está relacionada con los elementos de audio que componen el Módulo de Lógica. En el ejemplo, el nodo **“Bienvenido”** tiene un único elemento de audio de tipo **TEXTO** que mediante un conversor texto/voz se presentará al llamante como mensaje de bienvenida del Servicio 90X.

**Figura 6-9: Ejemplo información Nodo de tipo Locución**

**3.2.2. Información del nodo 2**

- Tipo de nodo: **nodo\_locucion**
- Nombre de nodo: Bienvenido
- Descripción: *"Locución de bienvenida del plan"*
- Transición: *EXITO*
- ¿Puede el usuario interrumpir la locución?: No
- Número de elementos de la locución: 1
- \*\*\*\*\* Elemento 1 \*\*\*\*\*
- Tipo: TEXTO
- Valor: "Bienvenido a la consulta del horóscopo."

#### 6.4.7 Análisis de los itinerarios sin salida válida del Plan

Esta sección es opcional y puede contener todos los itinerarios de esta clase o un subconjunto de ellos, en función de lo escogido por el usuario en la interfaz gráfica. Los itinerarios sin salida válida se caracterizan por contener, al menos, un nodo en el que se produce un evento de tipo *“no-input”* ó *“no-match”*, de manera que la llamada evoluciona hacia nodos alternativos en el Plan de Abonado. En la siguiente

imagen vemos un ejemplo de un itinerario de este tipo. En los itinerarios sin salida válida se marcará de color azul el nombre del nodo hacia el que evoluciona la llamada una vez producido el evento. En el ejemplo, vemos como el nodo de tipo **dato\_entrada** llamado *algomas*, provoca una transición sin salida válida hacia un nodo de tipo **Locucion** denominado *Max\_intentos*, en el que se informará al usuario de la situación.

**Figura 6-10: Ejemplo itinerario sin salida válida del Plan**



#### 6.4.8 Análisis de los itinerarios con errores del Plan

Los itinerarios con errores del Plan de Abonado, se caracterizan por contener, al menos, un nodo cuya transición al siguiente nodo es una transición de error, representada por una flecha de color rojo. Esta sección, tendrá también carácter

opcional, y podrá contener todos los itinerarios de error o sólo los escogidos por el usuario al comienzo de la aplicación. Los motivos por los que un determinado nodo dentro del Plan puede sufrir un error y provocar una transición de este tipo son muy numerosos.

En la siguiente figura, se representa un ejemplo de itinerario en el que un nodo de tipo **invocar\_ServicioExterno** sufre un error, motivado por ejemplo por una caída de la conexión. El siguiente nodo al que evoluciona la llamada tras el fallo, tendrá su nombre en rojo en la tabla del itinerario y será el primero que afronte y trate el error. Para este caso concreto, la llamada tras el fallo vuelve al nodo de tipo **Locucion** llamado *Bienvenida*, que consistirá en un audio inicial de presentación del Servicio 90X.

**Figura 6-11: Ejemplo de itinerario con errores del Plan**



#### 6.4.9 Análisis de los itinerarios con eventos de marcha atrás en el Plan

Esta sección es opcional, y al igual que las tres anteriores contiene información detallada de todos, o de alguno de los itinerarios con eventos de marcha atrás en el Plan. Dichos eventos se producen cuando el usuario, de forma voluntaria, quiere

volver sobre un nodo anterior en la lógica preestablecida del servicio. Por ejemplo, cuando queremos escuchar de nuevo una locución o repetir parte del plan con opciones diferentes. Las transiciones de vuelta atrás se caracterizarán con flechas verdes en el esquema del Plan desarrollado en el AFABLE IDE y se señalarán en la herramienta de documentación, marcando de color verde el nombre del primer nodo que cursa la llamada una vez producido el evento.

En la siguiente figura, se muestra un ejemplo de itinerario con eventos de marcha atrás. Cuando la llamada cursa el Módulo de Lógica representado por el nodo “*algomas*”, el usuario puede volver a escuchar la locución que le pide el dato de entrada eligiendo la opción **ATRÁS**.

**Figura 6-12: Ejemplo itinerario con eventos de marcha atrás en el Plan**





#### 6.4.10 Estudio de los Recursos del Plan

Esta sección es obligatoria y estará presente en todos los documentos generados por la aplicación. Consta de un primer párrafo explicativo en el que se describen las características principales de los cuatro tipos de recursos existentes en todo Plan de Abonado creado con el Servicio AFABLE.

- **Recursos de tipo Bases de Datos:** Son aquellas bases de datos que define el usuario directamente sobre el AFABLE IDE. La documentación recogerá, por cada recurso de este tipo los siguientes datos:
  - Nombre de la tabla
  - Descripción de la tabla (opcional)
  - Esquema de la tabla

**Figura 6-13: Ejemplo Recurso tipo Base de Datos**

ESQUEMA DE LA TABLA		
CAMPO	TIPO	ES_CLAVE
SIGNO	CADENA	SI
AMOR	CADENA	NO
SALUD	CADENA	NO
TRABAJO	CADENA	NO
DINERO	CADENA	NO
NUMERO_SUERTE	ENTERO	NO
PALABRA_CLAVE	CADENA	NO
PREDICCION	CADENA	NO

- **Recursos de tipo Servicio Externo:** Son aquéllos recursos en los que el usuario define en el AFABLE IDE el nombre y la URL de un servicio externo cuya misión es la de ejecutar una determinada acción y devolver una

respuesta en un formato concreto, entendible por el Sistema AFABLE y a partir de la cual se pueda cursar una u otra lógica en el Plan de Abonado. La información recogida en el documento para cada recurso de este tipo es la siguiente:

- Nombre del servicio
- Descripción del servicio (opcional)
- Url del servicio
- Esquema de la Petición al servicio externo y ejemplo de uso

**Figura 6-14: Ejemplo Recurso tipo Servicio Externo. PETICIÓN**

**PETICIÓN**

CAMPOS DE LA PETICIÓN: cz	
ZODIACO	CADENA

Ejemplo de uso de la petición:

<http://ubuntuwiki:14000/AgenteHoroscopo/servlet/horoscopo?OPERACION=cz&ZODIACO=XXX>

- Esquema de la Respuesta del servicio externo

**Figura 6-15: Ejemplo Recurso tipo Servicio Externo. RESPUESTA****RESPUESTA**

<b>CAMPOS DE LA RESPUESTA</b>	
<b>zodiaco</b>	CADENA
<b>texto</b>	CADENA
<b>textocorto</b>	CADENA
<b>amor</b>	CADENA
<b>salud</b>	CADENA
<b>trabajo</b>	CADENA
<b>dinero</b>	CADENA
<b>numero_suerte</b>	NUMERO
<b>pclave</b>	CADENA

- Ejemplo del código xml de respuesta que espera el Sistema AFABLE tras la ejecución del Servicio Externo

**Figura 6-16: Ejemplo de código xml de Respuesta de un Servicio Externo**

Ejemplo de código xml de la respuesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<respuesta>
  <registro>
    <campo>
      <nombre>zodiaco</nombre>
      <valor>XXX</valor>
    </campo>
    <campo>
      <nombre>texto</nombre>
      <valor>XXX</valor>
    </campo>
    <campo>
      <nombre>textocorto</nombre>
      <valor>XXX</valor>
    </campo>
    <campo>
      <nombre>amor</nombre>
      <valor>XXX</valor>
    </campo>
    <campo>
      <nombre>salud</nombre>
      <valor>XXX</valor>
    </campo>
    <campo>
      <nombre>trabajo</nombre>
      <valor>XXX</valor>
    </campo>
    <campo>
      <nombre>dinero</nombre>
      <valor>XXX</valor>
    </campo>
    <campo>
      <nombre>numero_suerte</nombre>
      <valor>XXX</valor>
    </campo>
    <campo>
      <nombre>pclave</nombre>
      <valor>XXX</valor>
    </campo>
  </registro>
</respuesta>
```

- **Ficheros de recursos:** Son aquéllos que el usuario adjunta directamente al Plan de Abonado. Son ficheros adjuntos que nada tienen que ver con la lógica empleada por el Sistema AFABLE. Pueden ser ficheros de audio, ficheros de gramáticas, rutinas EcmaScript, etc. La información recogida en el documento para este tipo de recursos es la siguiente:
  - Nombre del recurso

- Tipo del recurso
  - Subtipo del recurso
  - Origen del recurso
  - Ruta del recurso
- **Recursos de tipo Gramática:** Son aquéllas gramáticas que define el usuario “online” directamente sobre la interfaz gráfica del AFABLE IDE. La información recogida en el documento para este tipo de recursos es la siguiente:
- Nombre de la gramática
  - Descripción de la gramática (opcional)
  - Modo de entrada de la gramática
  - Número de vocablos de la gramática
  - Esquema de la gramática

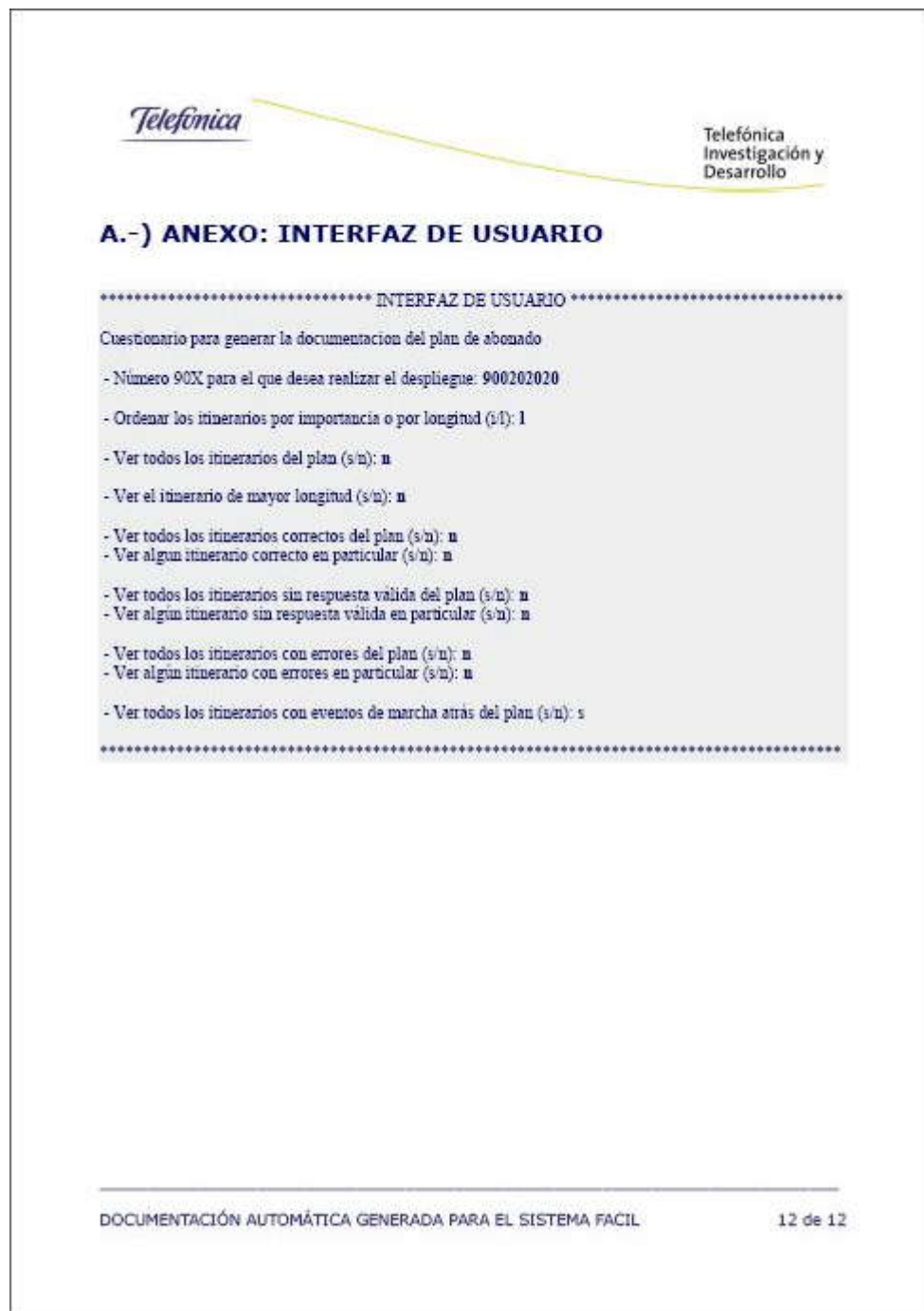
**Figura 6-17: Ejemplo Recurso tipo Gramática**

<b>ESQUEMA DE LA GRAMÁTICA</b>	
Aries	Aries
Tauro	Tauro
Geminis	Geminis
Cancer	Cancer
Leo	Leo
Virgo	Virgo
Libra	Libra
Escorpio	Escorpio
Sagitario	Sagitario
Capricornio	Capricornio
Acuario	Acuario
Piscis	Piscis

#### 6.4.11 ANEXO: Interfaz de Usuario

Es la última sección del documento del Plan de Abonado. Tiene carácter obligatorio y su objetivo es recoger las respuestas introducidas por el usuario en la interfaz gráfica con la que arranca la aplicación. Esta información permite al usuario comprobar de forma resumida los contenidos de los que consta el documento y le puede servir de guía para elaborar otros con distintas opciones.

Figura 6-18: Anexo Interfaz de Usuario



## 6.5 CONTROL DE ERRORES Y EXCEPCIONES

El control de errores y excepciones se ha concentrado en la clase **AfableDocException** contenida en el paquete *documentacion.exceptions*. Cualquier error que se produce en el programa es lanzado hacia atrás en la pila de llamadas hasta que se recoge y se trata por medio de este objeto. De esta forma, se consigue tener unificado el control de errores. Como veíamos en el diagrama de secuencia de primer nivel **4.5.1 Diagrama de secuencia de primer nivel**, es la clase **AfableDocumentacion** la que recoge las excepciones producidas por el sistema y las lanza hacia atrás con la información del mensaje de error.

El resultado de la aplicación cuando se produce una excepción se muestra a través de una ventana de diálogo en la que se indica que no ha sido posible generar la documentación del Plan. Algunos errores que se pueden producir son los siguientes:

- Se introduce un fichero del plan que no tiene extensión *.axml*.
- Se introduce un fichero de imagen para el plan con una extensión que no es *.jpg* ni *.png*.
- El fichero del plan no está bien validado o tiene errores.

## 6.6 SISTEMA DE TRAZAS

Para el seguimiento de trazas del sistema se ha utilizado la librería *log4j.jar*. **Log4j** es una biblioteca open source desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o “logs” en tiempo de ejecución y no en tiempo de compilación como suele ser habitual. La configuración de salida y granularidad de los mensajes es realizada en tiempo de ejecución mediante el uso de archivos de configuración externos. Se puede consultar más información en la bibliografía [2 **The Complete Manual log4j**].



### 6.6.1 Niveles de prioridad de los mensajes.

Por defecto Log4j tiene 6 niveles de prioridad para los mensajes (trace, debug, info, warn, error, fatal). Además existen otros dos niveles extras (all y off):

- **FATAL**: se utiliza para mensajes críticos del sistema, generalmente después de guardar el mensaje el programa abortará.
- **ERROR**: se utiliza en mensajes de error de la aplicación. Estos eventos afectan al programa pero lo dejan seguir funcionando, como por ejemplo que algún parámetro de configuración no es correcto y se carga el parámetro por defecto.
- **WARN**: se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan al correcto funcionamiento del programa.
- **INFO**: se utiliza para mensajes de información relativamente importantes, como por ejemplo, para avisar de la finalización de tareas complejas.
- **DEBUG**: se utiliza para escribir mensajes de depuración. Este nivel no debe estar activado cuando la aplicación se encuentre en producción.
- **TRACE**: se utiliza para mostrar mensajes con un mayor nivel de detalle que debug.

Respecto a los dos niveles extras:

- **ALL**: este es el nivel de máximo detalle, habilita todos los logs (en general equivale a TRACE).
- **OFF**: este es el nivel de mínimo detalle, deshabilita todos los logs.

### 6.6.2 Appenders

En **Log4J** los mensajes son enviados a una (o varias) salida de destino, lo que se denomina un *appender*.

Existen varios appenders disponibles y configurados, aunque también podemos crear y configurar nuestros propios appenders:

- **FileAppender** ó **RollingFileAppender**: La salida de los mensajes es redirigida a un fichero de texto .log. La diferencia entre los dos es que el segundo tiene carácter circular y cuando alcanza el máximo de su capacidad las nuevas trazas se van insertando a la vez que las primeras se eliminan.
- **SocketAppender**: La salida de los mensajes es redirigida a un servidor remoto donde se almacenan los registros.
- **SmtpAppender**: Se redirigen los mensajes a una dirección de correo electrónico.
- **JDBCAppender**: También es posible redirigir los mensajes a una base de datos.
- **ConsoleAppender**: Cuando se desea que los mensajes aparezcan en la salida estándar, salida de consola. Este tipo de appender no suele utilizarse en entornos de producción.

### 6.6.3 Layouts

En Log4j los layouts son los encargados de dar un formato de presentación a los mensajes. Se distinguen los siguientes tipos:

- **SimpleLayout** y **PatternLayout**. Permite presentar el mensaje con el formato necesario para almacenarlo simplemente en un archivo de texto.
- **HTMLLayout**. Presenta el mensaje en una tabla HTML.
- **XMLLayout**. Presenta el mensaje en un fichero XML

Además, es posible añadir información extra a cada uno de los mensajes, como puede ser la clase que generó el mensaje, la fecha en que se generó, etc.

### 6.6.4 Fichero log4j\_DOCUMENTACION.properties

El fichero de configuración de **Log4j** empleado en la aplicación se recoge a continuación:

```
# Coloca el nivel del logger en DEBUG y añade 4 appender
log4j.logger.documentacion=DEBUG, documentacionLOG, documentacionERR
log4j.logger.odf.plantilla=DEBUG, documentacionLOG, documentacionERR
log4j.logger.afable.documentacion.generador=DEBUG, generadorLOG,
generadorERR

#####documentacionLOG#####
log4j.appender.documentacionLOG=org.apache.log4j.RollingFileAppender
log4j.appender.documentacionLOG.Threshold=DEBUG
log4j.appender.documentacionLOG.MaxFileSize=10MB
log4j.appender.documentacionLOG.MaxBackupIndex=1
log4j.appender.documentacionLOG.layout=org.apache.log4j.PatternLayout
log4j.appender.documentacionLOG.File=DocumentacionAutomatica/Configuracion/Logs/documentacion.log
log4j.appender.documentacionLOG.layout.ConversionPattern=%d %-5p
[%t] - %C{1}: %m%n

#####documentacionERR#####
log4j.appender.documentacionERR=org.apache.log4j.RollingFileAppender
log4j.appender.documentacionERR.Threshold=ERROR
log4j.appender.documentacionERR.MaxFileSize=2MB
log4j.appender.documentacionERR.MaxBackupIndex=1
log4j.appender.documentacionERR.layout=org.apache.log4j.PatternLayout
log4j.appender.documentacionERR.File=DocumentacionAutomatica/Configuracion/Logs/documentacion.err
log4j.appender.documentacionERR.layout.ConversionPattern=%d %-5p
[%t] (%C.%M "%F:%L"). - %m%n

#####generadorLOG#####
```

```

log4j.appender.generadorLOG=org.apache.log4j.RollingFileAppender
log4j.appender.generadorLOG.Threshold=DEBUG
log4j.appender.generadorLOG.MaxFileSize=10MB
log4j.appender.generadorLOG.MaxBackupIndex=1
log4j.appender.generadorLOG.layout=org.apache.log4j.PatternLayout
log4j.appender.generadorLOG.File=DocumentacionAutomatica/Configuraci
on/Logs/generador.log
log4j.appender.generadorLOG.layout.ConversionPattern=%d %-5p [%t] -
%C{1}: %m%n

#####generadorERR#####
log4j.appender.generadorERR=org.apache.log4j.RollingFileAppender
log4j.appender.generadorERR.Threshold=ERROR
log4j.appender.generadorERR.MaxFileSize=2MB
log4j.appender.generadorERR.MaxBackupIndex=1
log4j.appender.generadorERR.layout=org.apache.log4j.PatternLayout
log4j.appender.generadorERR.File=DocumentacionAutomatica/Configuraci
on/Logs/generador.err
log4j.appender.generadorERR.layout.ConversionPattern=%d %-5p [%t]
(%C.%M "%F:%L") . - %m%n

```

Del fichero de configuración se pueden comentar los siguientes aspectos

- Existe un total de 4 appenders:
  - **documentacion.log**: Contiene todas las trazas de la aplicación de documentación a excepción de las relacionadas con el mapeo del fichero xml a objetos java.
  - **documentacion.err**: Contiene sólo las trazas de nivel de ERROR de documentación a excepción de las relacionadas con el mapeo del fichero xml a objetos java.
  - **generador.log**: Contiene las trazas del mapeo del fichero xml a objetos java.
  - **generador.err**: Contiene sólo las trazas de nivel de ERROR del mapeo del fichero xml a objetos java.

- Todos los appenders se inicializan con un nivel de **DEBUG** como nivel de prioridad por defecto.
- Todos los appenders son de tipo **RollingFileAppender**, por lo que las trazas se almacenan en fichero circulares donde no es necesario borrar trazas y las últimas reemplazan a las primeras a medida que se va llenando el archivo.
- El nivel de prioridad se puede ajustar para cada appender a partir de la propiedad **Threshold**. Sólo se visualizarán aquellos mensajes cuya prioridad sea mayor que la indicada en este parámetro.
- El layout utilizado para todos los fichero de trazas es **PatternLayout**, y añade información interesante a los mensajes, como puede ser:
  - **%d**: Fecha en la que produce el mensaje, con precisión de milisegundos.
  - **%-5p**: Nivel de prioridad en cinco caracteres.
  - **[%t]**: Entre corchetes, el hilo o proceso que genera el mensaje.
  - **%C{1}**: Clase que lanza el mensaje, la primera en la pila de llamadas. Se puede conseguir detalle de método **%M** o incluso de línea **%L**.
  - **%m%n**: El propio mensaje con un retorno de carro.

### 6.6.5 Forma de uso de Log4j

Para utilizar Log4j es necesario importar la librería log4j.jar a nuestro proyecto. A continuación es necesario seguir los siguientes pasos:

- Inicializar Log4j a partir de la ruta del fichero de propiedades que se ha visto en el apartado anterior. Sólo ha de realizarse una vez en el arranque de la aplicación.

```
PropertyConfigurator.configureAndWatch(rutaLog4j);
```

- En cada clase en la que se desee introducir mensajes de logs, es necesario crear una variable estática con el nombre de la clase que se va a escribir en el registro.

```
static Logger logger = Logger.getLogger(Configuracion.class);
```

- Utilizar los métodos `trace()`, `debug()`, `info()`, `warn()`, `error()`, etc, de la clase `Logger` para generar los mensajes.

## 6.7 PAQUETE DE LA APLICACIÓN

Uno de los requisitos funcionales de la herramienta, recogido en el apartado **3.2.1 Requisitos funcionales**, consistía en generar un fichero **.zip** con todos los elementos de la aplicación, de forma que el sistema fuera independiente y pudiera funcionar de forma autónoma en cualquier PC. Para conseguir esto se ha hecho uso de la herramienta **Ant**.

**Apache Ant [3]** es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas durante la fase de compilación de la aplicación. Hecha en lenguaje Java, tiene la ventaja de no depender del intérprete de comandos de cada Sistema Operativo, sino que se basa en ficheros de configuración XML y clases Java para la realización de las diferentes tareas, siendo una solución idónea en un escenario multiplataforma.

### 6.7.1 Fichero build.xml

El fichero **build.xml** permite configurar las tareas necesarias para generar el zip de la aplicación. Básicamente, este fichero de construcción planteará dos tareas:

1. “**Crear jar**”: Creará un jar ejecutable con todos los ficheros fuente de la aplicación, indicando la clase que contiene el método `main`. Dicho jar será el que se utilice para la integración de la herramienta en el AFABLE IDE.

2. “**Crear zip**”: Creará un fichero zip de nombre AFABLE DOCUMENTACION, que contendrá los ficheros de recursos de la aplicación, las librerías de clases y el jar ejecutable creado en el punto anterior. Contendrá también un fichero de extensión .bat con la instrucción a ejecutar para que arranque la aplicación.

El contenido del fichero se detalla a continuación:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<project name="AFABLE_DOCUMENTACION" default="AFABLE_DOCUMENTACION"
basedir=".">

    <property file="build.properties"/>

    <!-- #####-->
    <!-- Compilación y despliegue de AFABLE_DOCUMENTACION -->
    <!-- #####-->

    <target name="AFABLE_DOCUMENTACION" depends="crea_jar, crea_zip"
        description="Realiza las acciones completas para generar
el zip con toda la herramienta de documentacion">

        </target>

    <!-- #####-->
    <!-- Crea jar AFABLE_DOCUMENTACION -->
    <!-- #####-->

    <target name="crea_jar" description="Genera el JAR de la
aplicacion">

        <echo message="Creando el jar ..."/>

        <delete file="${dir.raiz}${nombre.proyecto}.jar"
failonerror="false"/>

        <jar destfile="${dir.raiz}${nombre.proyecto}.jar"
basedir="${dir.bin}" compress="true" >

            <manifest>

                <attribute name="Main-Class"
value="MainAfableDocumentacion"/>

            </manifest>

        </jar>

    </target>
```

---

```

<!-- #####-->
<!-- Crea zip completo AFABLE_DOCUMENTACION -->
<!-- #####-->

<target name="crea_zip" description="Genera el ZIP completo de la
aplicacion">

    <echo message="Creando el zip ..."/>

    <delete file="${dir.raiz}${nombre.proyecto}.zip"
failonerror="false"/>

    <zip destfile="${dir.raiz}${nombre.proyecto}.zip" >

        <zipfileset dir="${dir.documentacionAutomatica}"
prefix="DocumentacionAutomatica"/>

        <zipfileset file="${dir.raiz}${nombre.proyecto}.bat"/>

        <zipfileset file="${dir.lib}*.jar"/>

        <zipfileset dir="${dir.raiz}"
includes="${nombre.proyecto}.jar"
fullpath="${nombre.proyecto}.jar"/>

    </zip>

    <!--<delete file="${dir.raiz}${nombre.proyecto}.jar"
failonerror="false"/>-->

</target>
</project>

```

### 6.7.2 AFABLE\_DOCUMENTACION.zip

El fichero **AFABLE\_DOCUMENTACION.zip** contiene la siguiente estructura de ficheros y directorios:

- **DocumentaciónAutomática.** Contiene la información relacionada con los ficheros de recursos, imágenes de iconos y logs de la aplicación. Está formado a su vez por tres directorios:
  - **Configuración:** Directorio que contiene el fichero de propiedades de Log4j para las trazas, además del directorio **Logs** donde se guardarán los ficheros de trazas de los cuatro appenders que se vieron en el apartado **6.6.4 Fichero log4j\_DOCUMENTACION.properties.**



- **IconosNodos:** Guarda las imágenes y los logos necesarios para generar el Documento Automático de la aplicación.
- **Recursos:** Contiene los ficheros .xml de recursos de la aplicación. Estos ficheros son los siguientes:
  - \* configuracion.xml: Contiene datos importantes de configuración como la ruta del fichero de properties para las trazas o la propia ruta de los ficheros de recursos.
  - \* constantes.xml: Contiene parejas de clave-valor para identificar constantes que se utilizan en el proyecto.
  - \* iconos.xml: Contiene las etiquetas que identifican cada uno de los ficheros de iconos que se utilizan.
  - \* mensajes.xml: Contiene la información de los mensajes de información a presentar al usuario en cada situación.
  - \* preguntas.xml: Contiene las opciones de la interfaz gráfica de usuario.
  - \* secciones.xml: Contiene los títulos de las secciones del documento, así como el contenido de los párrafos explicativos de cada sección.
- Fichero **AFABLE\_DOCUMENTACION.jar**: Contiene todos los fuentes de la aplicación, tal y como se ha explicado en el apartado anterior.
- **Ficheros de librerías:** Se añaden al paquete zip, todas las librerías que utiliza el proyecto. Dichas librerías son las siguientes:
  - **log4j.jar**: Necesaria para la implementación de trazas.
  - **odfdom.jar**: Contiene las clases del API ODFDOM, versión 0.6.16.
  - **xercesImpl.jar**: Se utiliza para el mapeo de ficheros xml a objetos Java.
  - **swing-layout-1.03.jar**: Se utiliza para desarrollos gráficos con JSwing.

- *managerAFABLEDocumentacion.jar*: Contiene un subconjunto de clases del AFABLE MANAGER necesarias para la documentación.
- *generadorAFABLEDocumentacion.jar*: Contiene un subconjunto de clases del AFABLE SERVER necesarias para la documentación.
- Fichero ejecutable **AFABLE\_DOCUMENTACION.bat**. Contiene la instrucción que arranca la aplicación de forma autónoma. Para su correcto funcionamiento, es necesario primero descomprimir todo el paquete zip en un directorio.

```
@echo off  
  
java -classpath  
log4j.jar;managerAFABLEDocumentacion.jar;generadorAFABLEDocum  
entacion.jar;odfdom.jar;xercesImpl.jar;swing-layout-  
1.0.3.jar;AFABLE_DOCUMENTACION.jar;. MainAfableDocumentacion
```

## 6.8 INTEGRACIÓN EN EL AFABLE IDE

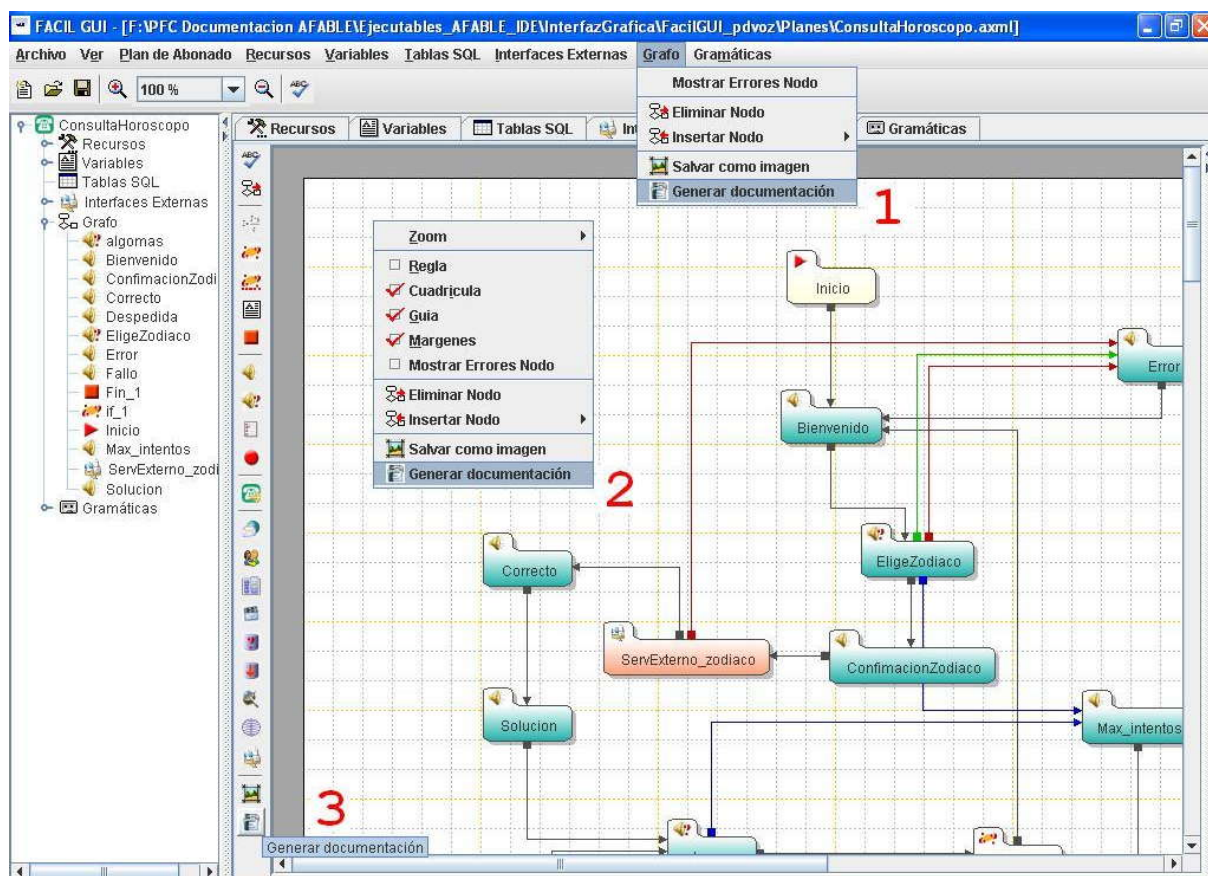
Uno de los requisitos funcionales de la aplicación, recogido en el tercer punto del apartado **3.2.1 Requisitos funcionales**, consistía en ser capaces de integrar el módulo **AFABLE DOCUMENTACIÓN** desarrollado en el presente proyecto dentro del Sistema AFABLE y más concretamente, dentro de la herramienta gráfica de edición de planes denominada AFABLE IDE, cuyas características principales se detallan en el anexo **A.2.1 AFABLE IDE**.

En la siguiente figura se muestra la integración de la herramienta de documentación dentro del interfaz gráfico del AFABLE IDE. Existen, como se ve en la imagen, hasta tres puntos de acceso:

1. Dentro de la barra de menús, seleccionando la opción **“Grafo”**
2. Haciendo click con el botón derecho del ratón sobre un lugar vacío de la ventana de edición del grafo

- Dentro de la barra vertical de botones, que presenta los “atajos” de las diversas opciones de la barra de menús.

**Figura 6-19: Integración en el AFABLE IDE**



Una vez que se selecciona la opción “**Generar documentación**”, el sistema nos preguntará si deseamos o no incluir la imagen del Plan. Si respondemos afirmativamente, se abrirá una ventana de selección de ficheros donde deberemos escoger la imagen del Plan. Si por el contrario, respondemos que no, el apartado con la imagen del Plan no se incluirá dentro del Documento Automático generado por la aplicación. Una buena idea para conseguir la imagen del grafo en nuestro PC es utilizar la opción “**Salvar como imagen**” del AFABLE IDE. Dicha opción, presenta la imagen del grafo dentro del layout en el que está creado, por lo que puede ser necesario tratarla posteriormente antes de incluirla en el documento.

## 6.9 PRUEBAS DEL SISTEMA

Las pruebas son la fase final del ciclo de desarrollo, sin embargo, debido al modelo de desarrollo elegido (*cascada con retroceso o retroalimentación*) se han realizado pruebas en todas las fases del diseño. Una correcta batería de pruebas resulta fundamental para medir la calidad del software desarrollado (**verificación**) así como la medida en que cumple con los requisitos especificados (**validación**).

Tal y como se recoge en [3 Jose A. Mañas “Prueba de Programas”] existen varios tipos de pruebas, en función del momento en el que se encuentre el desarrollo y los objetivos que se persigan. Las más importantes son:

- **Pruebas unitarias:** Es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. En este sentido se distinguieron los siguientes módulos lógicos en el desarrollo.
  - *Mapeo de ficheros xml a objetos Java.* Se probó con decenas de ficheros xml de Planes de Abonado, comprobando que la generación de los objetos se realizaba correctamente.
  - *Aplicación del Algoritmo de Ordenación de Nodos.* Se probó con diferentes Planes que el algoritmo funcionaba y la generación de caminos se realizaba correctamente.
  - *Sistemas antibucles.* Se probó la técnica antibucles basada en árboles lógicos comprobando que no se entraban en métodos recursivos de ejecución infinita.
  - *Generación de documentos sencillos.* Se probaron los estilos poco a poco, sin mezclar lógica ni plantear referencias a contenido.
- **Pruebas de integración:** Pruebas de verificación y validación realizadas para comprobar el correcto funcionamiento de varios módulos de lógica una vez que se unen entre sí. En el desarrollo de la aplicación se procedió al acoplamiento de los diferentes módulos de lógica realizando pruebas

jerárquicas, de modo que se pudieran probar por separado los dos grandes bloques de la herramienta:

- *Generación de los itinerarios del Plan de Abonado.*
- *Generación de un documento en estándar abierto con encabezados, párrafos, listas, tablas y otros elementos de estilo.*

Las últimas pruebas de integración tuvieron lugar en el momento en el que se incorporó la lógica del primer bloque a la estructura del documento desarrollando en el segundo.

- **Pruebas de caja blanca:** Son aquéllas que se realizan sobre las funciones internas del código. En ellas, estamos siempre observando el código. En ocasiones, se denominan también pruebas de cobertura puesto que su misión es probar satisfactoriamente la mayor porción de código posible. En las pruebas de caja blanca, resultan especialmente importantes las trazas del sistema, puesto que nos permite ver el flujo de ejecución y detectar, en caso de errores, la porción de código que es necesario sustituir.
- **Pruebas de caja negra:** Son aquéllas que se centran en lo que se espera de un determinado módulo. No somos conscientes del código que se ejecuta por debajo, simplemente proporcionamos al módulo de lógica las entradas necesarias y miramos si la salida es satisfactoria. Suelen ser muy importantes, una vez que se realiza la integración de varios módulos lógicos y una buena práctica antes de empezar con pruebas de mayor detalle.

En la aplicación, se desarrollaron pruebas de caja negra para ver la generación de documentos y para probar posibles estados de fallo, como pueden ser la introducción de parámetros de entrada incorrectos.

- **Pruebas de carga:** Son pruebas destinadas a medir el rendimiento del desarrollo software cuando se le expone a situaciones extremas de carga. Para la realización de pruebas de carga se diseñaron planes específicos con un número muy elevado de nodos. Se probó que con planes con centenares

de nodos y más de cien itinerarios posibles el tiempo que se tardaba en generar la documentación apenas se incrementaba en unos segundos, a pesar de que el tamaño del fichero y el número de páginas del mismo crecía considerablemente.

---

## 7 CONCLUSIONES Y LÍNEAS FUTURAS DE DESARROLLO

### 7.1 CONCLUSIONES GENERALES

El módulo **AFABLE DOCUMENTACIÓN**, implementado en el presente Proyecto Fin de Carrera proporciona al Sistema AFABLE un complemento imprescindible para la documentación de los Planes de Abonado. A medida que los Planes de Abonado crecen y aumenta el número de Módulos de Lógica, no resulta trivial el discernir a simple vista los posibles caminos que puede seguir una llamada que cursa el servicio.

Los clientes del Sistema AFABLE, una vez confeccionaban un Plan de Abonado relativamente complejo, manifestaban su interés en los siguientes aspectos:

- o ¿Cuál es el camino más importante que puede cursar una llamada en todo el entramado de nodos del plan?
- o ¿Están capturados todos los posibles errores con los que se puede encontrar una llamada?
- o ¿Podría tener documentado todas las locuciones del plan?
- o ¿De todos los posibles caminos que puede efectuar la llamada, cuáles son los correctos?
- o ¿Podría tener documentado todos los recursos del Plan de Abonado?
- o ¿Podría conocer el detalle de todos los nodos que conforman un itinerario especialmente significativo?

Para responder a estas preguntas era necesario estudiar cada Plan de Abonado en concreto y elaborar manualmente un documento a medida de las peticiones del cliente. Además de resultar una labor tediosa, cuando el número de Módulos de

Lógica era muy alto, se convertía prácticamente en una tarea inviable, especialmente para personal ajeno a la confección del Plan en el AFABLE IDE.

La herramienta de Generación Automática de Documentación para el Sistema AFABLE ofrece una respuesta personalizada a estas y otras muchas preguntas. Permite generar documentos automáticos a medida de las necesidades del cliente, de forma rápida y eficaz. Sus principales ventajas se resumen en los siguientes puntos:

- El documento automático se genera con gran rapidez, incluso con Planes de Abonado que superan el centenar de nodos.
- La agrupación de caminos por colores permite una documentación clara y sencilla.
- La distinción del itinerario principal permite ver rápidamente el camino más importante por el que puede cursarse una llamada.
- Los criterios de ordenación permiten por un lado manejar los itinerarios más largos del Plan (ordenación por longitud) y por otro, permitir crear itinerarios de distintos pesos actuando sobre propiedades concretas de los nodos. Esto último posibilita manipular la importancia de los itinerarios para presentar en la documentación como itinerario principal uno creado por el propio usuario.
- La interfaz gráfica de la aplicación provoca que el documento automático sea totalmente personalizado, hasta el punto de que el usuario pueda elegir los itinerarios concretos que desea se incorporen al documento final. En este sentido, tienen especial importancia los listados de caminos que se generan en la aplicación, puesto que permiten reconocer de forma rápida el número de itinerario que se corresponde con una secuencia determinada de nodos e incluir dicho camino en un documento posterior.
- Los Recursos del Plan quedan explicados detalladamente, incluso con ejemplos de aplicación. Cualquier cambio que se realice en una base de datos, una gramática o un fichero de recursos queda contemplado en la documentación.



- El anexo con la Interfaz de Usuario permite a los clientes saber las opciones concretas con las que está generado cada documento.

Una vez resueltas las necesidades de los clientes, otra prioridad se basaba en especificar que formato iba a tener esta documentación personalizada. Desde el punto de vista de los responsables del Sistema AFABLE, parecía muy interesante hacer coincidir en la medida de lo posible, la estructura y el diseño del documento automático con la de cualquier otro documento generado de forma manual para describir un determinado proyecto software. Con esto, no sólo conseguían mantener un diseño corporativo del documento sino que podían entregar a los clientes una documentación con cuyo formato se encontraban ya familiarizados

En este sentido, ha sido necesario un gran esfuerzo de aprendizaje enfocado principalmente a dos frentes:

- Encontrar la estructura de datos adecuada que permitiera modelar el problema de los itinerarios del Plan de Abonado y sobre la que se pudiera resolver el problema de la repetición de nodos dentro de un mismo camino. Una estructura potente y flexible que permitiera una ordenación de nodos rápida y eficaz.
- Por otro lado, entender y aplicar la especificación de formato abierto definida por **OASIS** para **OpenDocument**. Cada uno de los elementos lógicos introducidos en el documento **.odt**, tiene una especificación formal en el estándar y una representación lógica no siempre inmediata en el **API ODFDOM**. La versión del API que se ha manejado, última versión hasta la fecha, contenía ciertas limitaciones y un vacío prácticamente absoluto en lo que se refiere a ejemplos de aplicación. Ha sido necesaria una profunda labor de estudio, de documentación y de participación en foros para extraer el rendimiento necesario y aplicarlo al documento automático generado por la aplicación.

## 7.2 FUTURAS MEJORAS Y ANÁLISIS CRÍTICO

El presente Proyecto Fin de Carrera plantea otra forma de entender la documentación de un proyecto software. Se trata de integrar el proceso de documentación dentro del propio desarrollo formando parte de la lógica de negocio. Con esto se conseguiría, por una parte, elevar el concepto y la importancia de la documentación de usuario hasta los términos que merece y por otro, poder dinamizar el contenido de los documentos de forma que se ajusten a las necesidades de los clientes que utilicen la herramienta.

En el ámbito de la presente aplicación, se han pensado una serie de futuras mejoras cuyo resumen se presenta a continuación:

- Podría resultar interesante ampliar la interfaz gráfica de forma que el usuario pudiera elegir caminos seleccionando el nombre de los nodos concretos que desea que existan dentro de cada itinerario. Una vez integrada dentro del AFABLE IDE, esta selección de nodos se podría hacer directamente sobre la ventana de edición del grafo.
- Futuras versiones podrían contemplar la posibilidad de generar varios documentos de forma simultánea, sin necesidad de ejecutar la aplicación varias veces.
- Las posibilidades del **API ODFDOM** y la fiabilidad del estándar **OpenDocument** podrían extender el concepto del documento automático más allá del formato de texto. Futuras versiones de la aplicación podrían plantear ficheros de presentación **.odp**, más vistosos, con multitud de efectos, convirtiendo cada documento de Plan de Abonado en una demo interactiva y personalizada.

Además, las futuras revisiones de la aplicación, deberán mejorar ciertos aspectos y limitaciones detectadas en el presente Proyecto. Entre ellas podemos destacar las siguientes:

- La creación de Planes de Abonado con un elevado número de itinerarios no supone un aumento en el tiempo de ejecución de la aplicación pero si que provoca, por lo general, que los documentos automáticos generados aumenten de dimensión considerablemente. No es un problema del tamaño de los ficheros, puesto que resultan claramente manejables, sino del número de páginas de los documentos. Los usuarios deben ser conscientes, y utilizar más que nunca en estos casos, la capacidad de la herramienta de generar documentos con itinerarios concretos.
- El estándar de documentación elegido requiere de la instalación de la suite ofimática de libre distribución **OpenOffice** para la visualización correcta de los documentos. Como se explicó en la sección **3.3.2.3 Estándar de documentación abierto OpenDocument (ODF)**, existen otras muchas aplicaciones y pluggins capaces de abrir ficheros **.odt**, sin embargo, resulta frecuente que existan pequeñas variaciones en la representación y visualización de ciertos elementos de estilo. La tendencia de cada una de estas aplicaciones es mejorar su capacidad para el tratamiento de este tipo de ficheros. Incluso estándares propietarios como Microsoft Office integran su plugin en la nueva versión 2007. En cualquier caso, la extensión y aceptación de OpenOffice como suite ofimática a tener en cuenta gana terreno poco a poco y empiezan a ser más las empresas y organismos oficiales, que en su apuesta por el software libre, presentan a OpenOffice como su paquete ofimático de referencia.
- Existe la posibilidad de utilizar la versión Portable de OpenOffice para la visualización de los documentos generados por la herramienta. Esta versión supone una ventaja clara respecto a la anterior y es que permite a los comerciales del Sistema AFABLE hacer una demostración de la herramienta de documentación sin tener en cuenta el software instalado en el PC del cliente, sin más que guardando la versión Portable en un simple lápiz de memoria. La versión Portable de OpenOffice tiene limitaciones de velocidad importante en dispositivos de memoria con una baja tasa de transferencia.

Esto puede provocar, que en soportes de memoria no demasiado potentes, la visualización y navegación por el documento se ralentice demasiado.

---

## A DESCRIPCIÓN GENERAL DEL SISTEMA AFABLE

### A.1 OBJETIVO

Las empresas que ofrecen **Números Novecientos (90X)** para la atención telefónica de sus clientes y/o empleados demandan facilidades más complejas que las que ofrecen actualmente de forma estándar, las empresas que desarrollan plataformas de Red Inteligente.

Para tratar de cubrir estas nuevas necesidades empresariales actualmente se está recurriendo a desarrollos a medida “fuera” de la propia Red Inteligente. Aunque estos desarrollos a medida consiguen satisfacer las necesidades de los clientes tienen inconvenientes importantes que en ocasiones pueden provocar incluso la pérdida de clientes:

- Plazo de disponibilidad
- Costes de desarrollo (capex)
- Costes de explotación (opex)

Un ejemplo de este tipo de servicios podría ser el siguiente: *“Una Universidad decide ofrecer la consulta de notas a través de un número 90X”*. Existen dos opciones para su puesta en marcha:

- Opción 1: Contrata con el operador telefónico un número 90X con dos terminaciones.
  - o Ventajas: El servicio se puede ofrecer en un corto periodo de tiempo
  - o Inconvenientes: La universidad tiene que disponer de teleoperadores para atender las llamadas telefónicas (en las dos terminaciones) con los consiguientes problemas de costes, horarios de atención, etc.
- Opción 2: Se desarrolla un servicio 90X automático (IVR).

- o Ventajas: La universidad no necesita tener teleoperadores
- o Inconvenientes: Hay que pagar unos desarrollos y esperar a que estos estén realizados. El coste de los desarrollos pueden ser no asumibles para la universidad.

Con el objetivo de resolver estos problemas se propone la realización del **sistema avanzado de Automatización de Funciones de Atención telefónica Basado en Lógicas Especiales (AFABLE)** que permita el diseño rápido y económico de servicios 90X personalizados sin necesidad de realizar complejos desarrollos software a medida.

El objetivo del sistema AFABLE es proporcionar un conjunto de herramientas para dar soporte al diseño, creación, despliegue y explotación de los Planes de Abonado VoiceXML de Red Inteligente y en general, a través de interfaces Web, a todas las fases del ciclo de vida de dichos servicios.

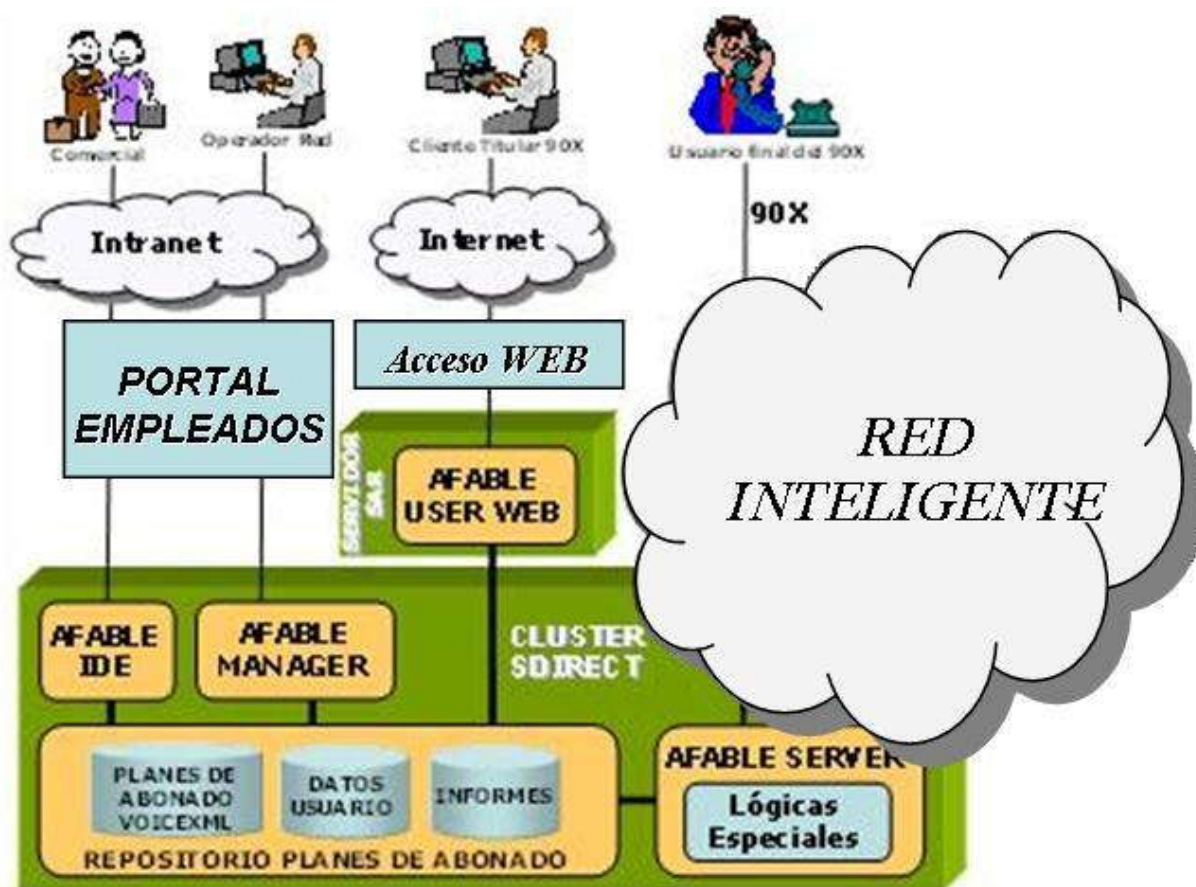
El Sistema AFABLE aporta los siguientes beneficios para el operador:

- Dispondrá de una herramienta avanzada para creación de servicios dotados de funciones avanzadas de atención telefónica.
- Podrá ofrecer a sus clientes que demandan 90X un servicio personalizado prácticamente de forma inmediata.
- Los costes de desarrollo no se “imputan” directamente a una empresa cliente pues las facilidades estarán disponibles para muchas empresas.
- Incluso en el futuro la propia empresa podrá configurar su servicio personalizado de una forma similar a como lo hace actualmente mediante la combinación de facilidades básicas.

## **A.2 COMPONENTES DEL SISTEMA AFABLE**

La figura representa gráficamente los diferentes componentes que intervienen en la creación y provisión de los Servicios VoiceXML Afable.

Figura A-1: Descripción general del Sistema AFABLE



En la figura se distinguen las cuatro herramientas software que forman el sistema AFABLE (IDE, MANAGER, SERVER y USER WEB).

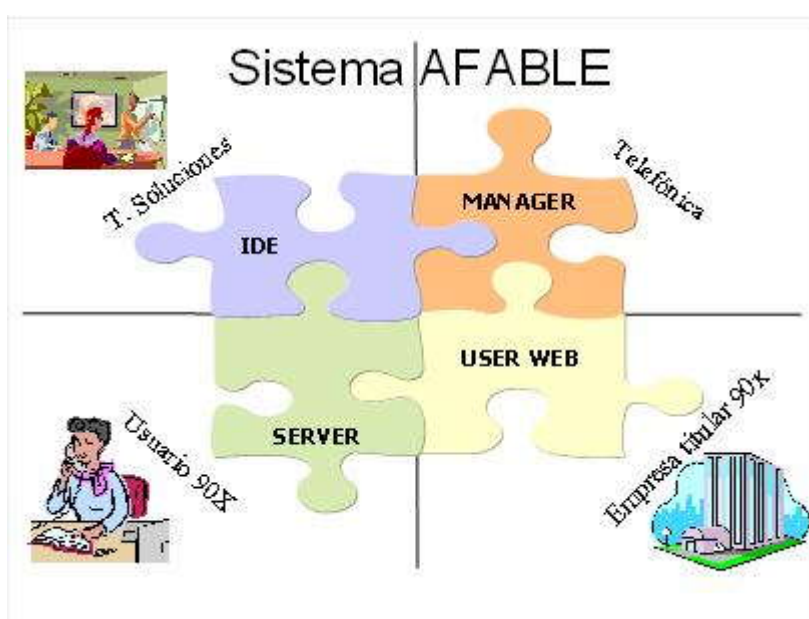
Estas herramientas comparten el REPOSITORIO DE PLANES DE ABONADO en el que se almacena de forma permanente (en ficheros y base de datos MySQL) toda la información de definición de los Planes de Abonado y datos de configuración y uso de las aplicaciones VoiceXML que se ejecutan en el sistema AFABLE SERVER. Desde este repositorio de datos se realiza la creación, despliegue, modificación y repliegue de los Planes de Abonado VoiceXML en los sistemas de operación de la Red Inteligente del operador.

Las herramientas IDE, MANAGER y USER WEB proporcionan servicios Web para la creación, administración y gestión de la explotación de los servicios 90X y sus

correspondientes planes de abonado. Como se muestra en la figura, cada herramienta está orientada a un perfil de usuario diferente: los Comerciales del operador, las unidades de tramitación y operación del operador e incluso los titulares de los servicios 90X (éstos últimos a través de la Web) pueden gestionar ciertos aspectos de los Planes de Abonado VoiceXML.

La figura siguiente muestra las relaciones entre el sistema AFABLE y sus usuarios.

**Figura A-2: Usuarios del Sistema AFABLE**



### A.2.1 AFABLE IDE

Es la herramienta Web para el diseño gráfico y creación de Planes de Abonado VoiceXML que ofrece las funciones siguientes de soporte al ciclo de vida de los Planes de Abonado a través de una interfaz Web gráfica:

- Creación
- Consulta
- Modificación



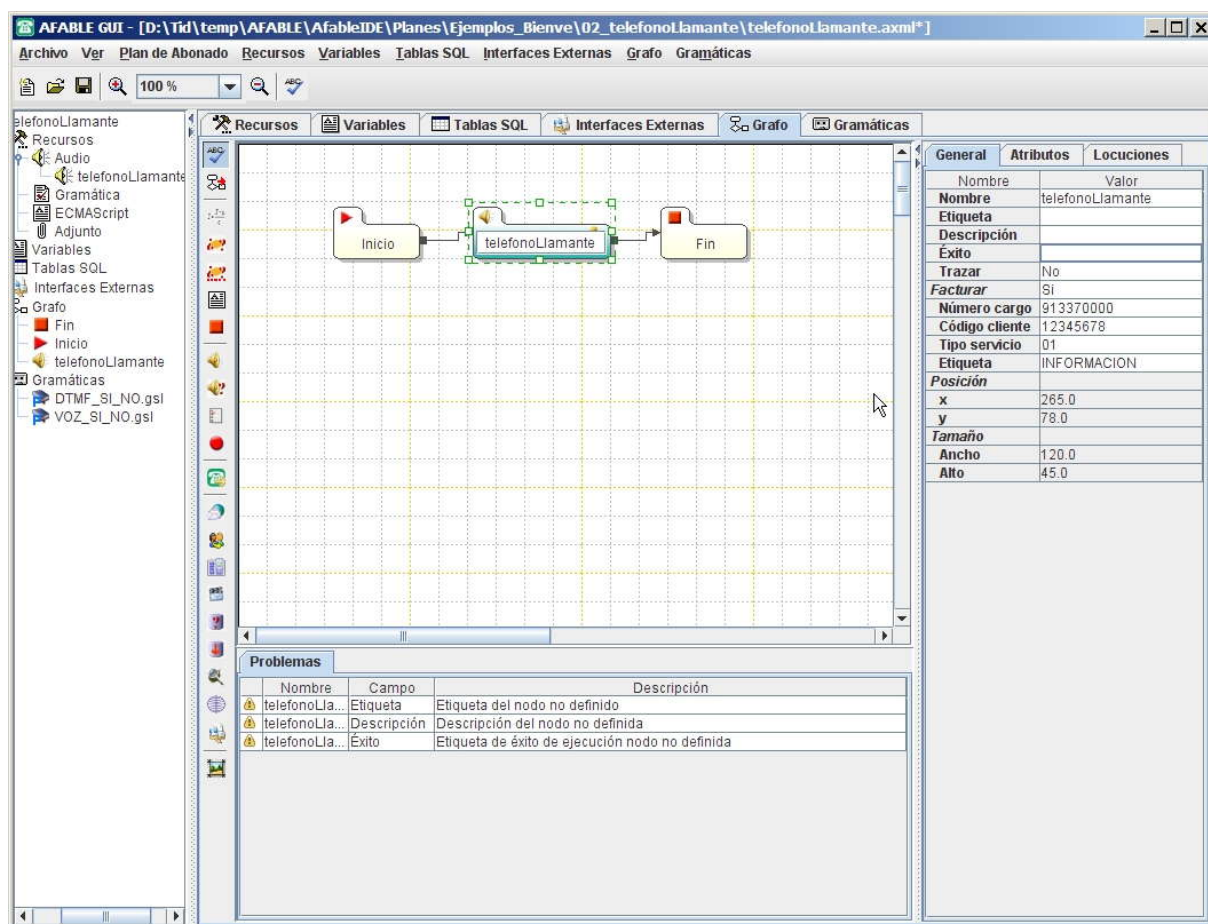
- Validación

Proporciona una interfaz gráfica guiada para el diseño rápido, completo y correcto de los servicios telefónicos personalizados. Para ello dispone de opciones gráficas que permiten definir el grafo del plan de servicio (módulos de lógica y transiciones entre éstos), así como el resto de elementos necesarios del servicio, tales como gramáticas, locuciones pregrabadas y algoritmos de programación (ECMAScript) que requiera cada servicio.

El conjunto de los módulos de lógica que proporciona AFABLE IDE incluye lógicas para definir las interacciones con el usuario, lógicas de programación (if, case, rutinas, etc.), lógicas de telefonía (transferir llamadas, colgar, etc.) y las Lógicas Especiales.

Las Lógicas Especiales constituyen una característica avanzada e innovadora ya que permiten añadir funcionalidad avanzada a los servicios telefónicos personalizados como son el acceso a base de datos, servicios Web, mensajería fija, móvil y e-mail, e incluso el acceso a otras fuentes de naturaleza similar a las utilizadas en el entorno Web.

Figura A-3: Ventana principal de AFABLE IDE



AFABLE IDE es una aplicación Java “*stand-alone*”, ligera que se ejecuta sobre una máquina virtual Java JRE 1.5 de Sun o compatible. La edición de los planes de servicio se realiza sobre el disco local del PC del diseñador sin necesidad de acceder a recursos externos al propio PC, es decir, sin necesidad de estar conectado a ninguna red. El plan de servicio se guarda de forma permanente en el disco en un Documento XML de acuerdo al esquema XML “*PlanAbonadoVoiceXML V1.0*”. AFABLE IDE únicamente se debe conectar a la red para realizar el envío y descarga de planes de servicio desde el Repositorio de Servicios AFABLE a través de los servicios Web que proporciona AFABLE MANAGER.

### **A.2.2 AFABLE MANAGER**

Proporciona funciones de Gestión de los Planes de Abonado VoiceXML en Red, orientadas al despliegue y repliegue de los servicios en producción. Junto con el componente AFABLE IDE, ofrece el conjunto de funciones necesarias para dar soporte completo al ciclo de vida de los Planes de Abonado a través de una interfaz Web. Estas operaciones son las siguientes:

- Generación automática de la Aplicación VoiceXML
- Despliegue
- Repliegue
- Activación
- Desactivación
- Eliminación

También ofrece un conjunto de funciones para el manejo de los datos de usuario de los servicios 90X que requieran de éstos, a través de una interfaz Web. Las operaciones de gestión de los datos de usuario son:

- Creación de Tabla de Datos
- Consulta de Tabla de Datos
- Alta de registro de datos
- Baja de registro de datos
- Carga masiva de registros de datos (reemplazo total o actualización parcial)
- Borrado de registros de datos

Y también ofrece funciones orientadas a la gestión de la explotación. Ofrece un conjunto de servicios Web para la operación monitorización y consulta de informes acerca del sistema del servicio:

- Limpieza y backups de logs y datos de operación
- Consulta de estadísticas e informes
- Gestión de los usuarios
- Gestión de la configuración
- Arranque y parada
- Monitorización del sistema:
  - Activar/desactivar las trazas para un Abono
  - Consulta del log de errores
  - Consulta del espacio en disco
  - Consulta del nivel de carga de la CPU

### **A.2.3 AFABLE SERVER**

Es el sistema de operación que sirve las páginas VoiceXML a los servidores de recursos de RI que ejecutan los Servicios 90X que atienden las llamadas de los usuarios.

Como parte del servidor AFABLE SERVER se encuentran las LÓGICAS ESPECIALES que permiten incorporar funcionalidad avanzada a los servicios 90X. Son funciones desarrolladas como parte del servidor de aplicaciones VoiceXML AFABLE SERVER, que permiten incorporar dicha funcionalidad a las páginas VoiceXML que implementan los Planes de Abonado VoiceXML. Estas lógicas especiales son:

- Envío de correo electrónico saliente

- Envío de mensaje al CAR obtenidos mediante conversión texto/voz o mediante una grabación de voz.
- Envío de SMS a móvil
- Envío de MMS a móvil
- Consulta de información de geolocalización del teléfono A
- Consulta de información sobre una base de datos local o remota
- Actualización de información sobre una base de datos local o remota
- Creación dinámica de gramáticas en tempo de ejecución del servicio

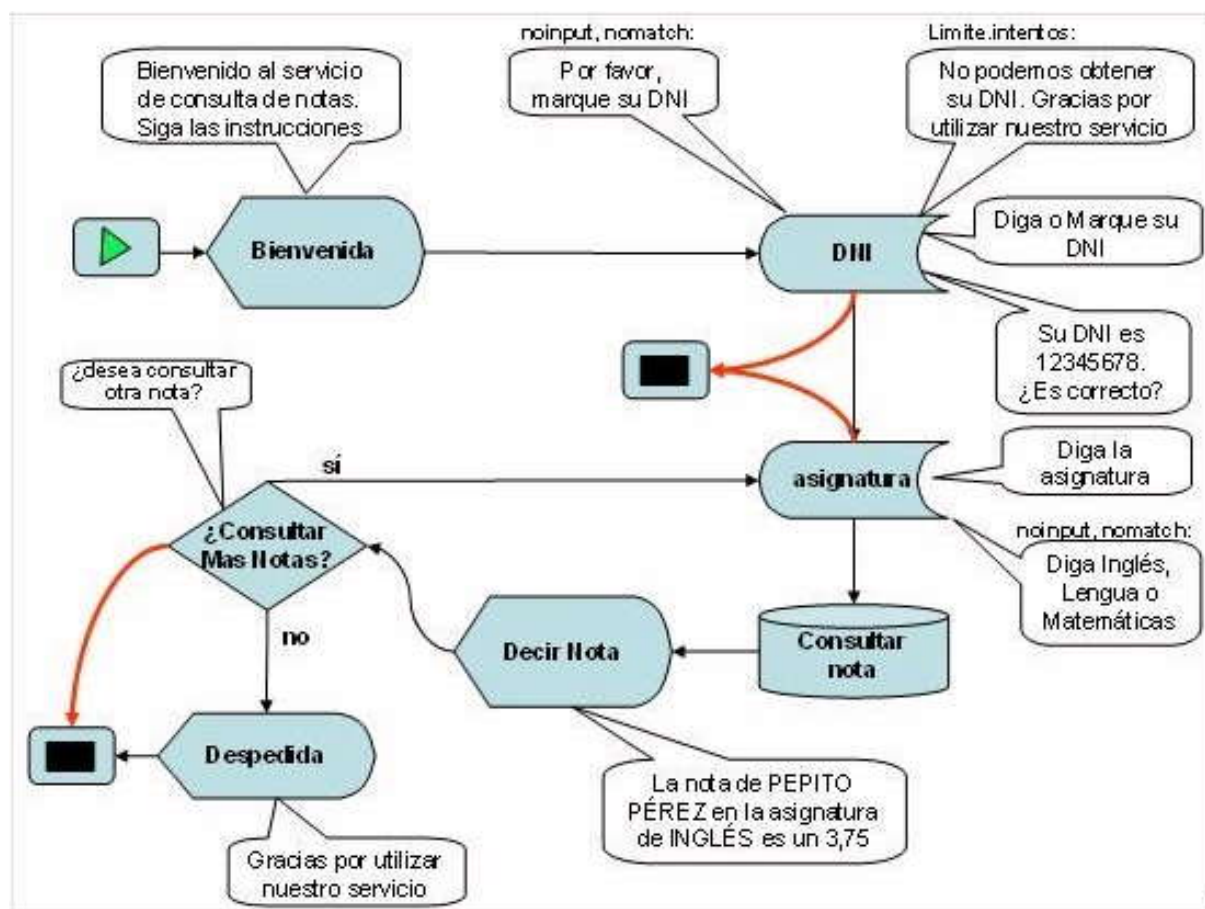
#### **A.2.4 AFABLE USER WEB**

Proporciona funciones de autogestión de los Planes de Abonado VoiceXML en Red a los Titulares de los servicios 90X, orientadas a la actualización de datos del servicio y consulta de informes de uso del mismo:

- Consulta de estadística e informes.
- Carga masiva de registros de datos de usuario (reemplazo total o actualización parcial)

### **A.3 EJEMPLO: “Servicio de consulta de notas”**

Este apartado presenta, a modo de ejemplo, un Plan de Abonado VoiceXML diseñado específicamente para proporcionar el servicio automático de “*consulta de las notas de un alumno identificado por su DNI a través de un 90X*”, con el fin de ilustrar toda la funcionalidad de la herramienta AFABLE.

**Figura A-4: Grafo del servicio de Consulta de Notas**

Las *flechas* representan las transiciones entre módulos de lógicas y los *bocadillos* representan las locuciones que oirá el usuario que interacciona con el servicio.

### A.3.1 Documento XML del Plan de Abonado

Un Plan de Abonado VoiceXML tiene una representación en formato XML de acuerdo a la estructura y contenido especificados en el esquema XML denominado **“XML Schema Plan Abonado VoiceXML V1.0”**

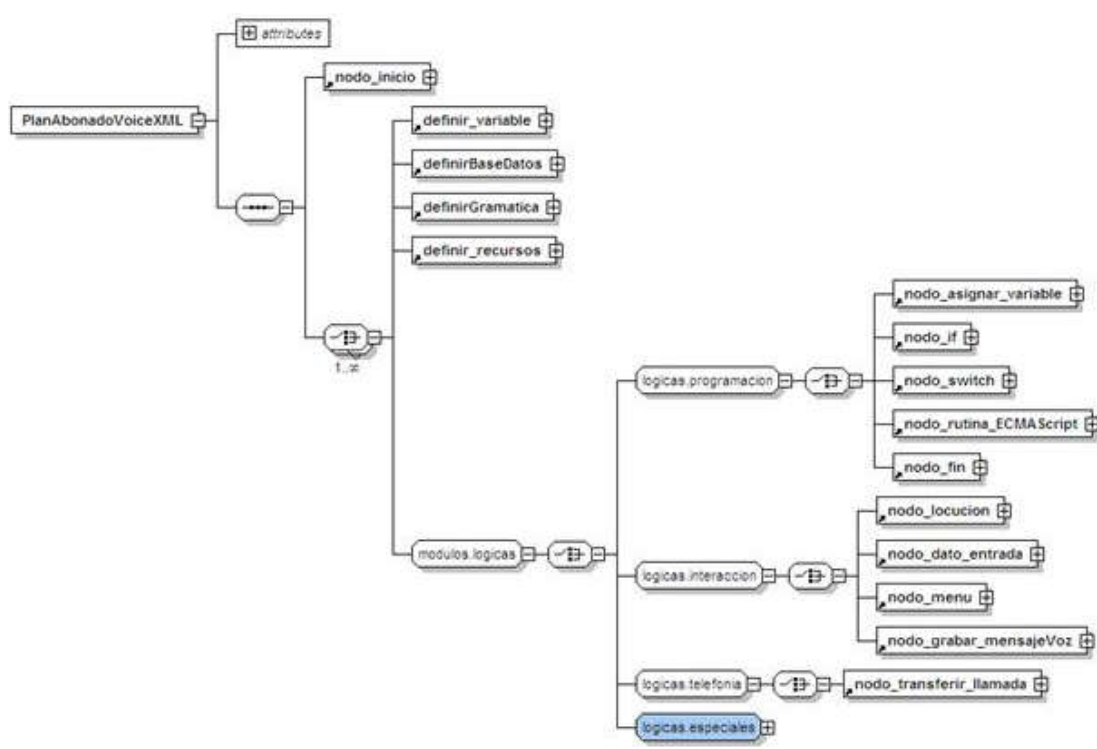
La edición del documento XML que define un plan de abonado se realiza mediante la herramienta AFABLE IDE.

Existirá una relación biunívoca entre la representación gráfica construida mediante la herramienta AFABLE IDE y el documento XML correspondiente.

Todo plan de abonado tiene un elemento principal denominado ***PlanAbonadoVoiceXML*** que lo define completamente. Dicho elemento principal define todos los elementos y la estructura funcional que determina la funcionalidad de cualquier Plan de Abonado VoiceXML.

La figura siguiente muestra la estructura del elemento principal ***PlanAbonadoVoiceXML***

**Figura A-5: XML Schema: PlanAbonadoVoiceXML**



Como se observa en la figura, el elemento ***PlanAbonadoVoiceXML*** está formado por unos atributos (***attributes***: identificación, autor, fecha de creación y versión – no se muestran en pantalla -), un ***nodo\_inicio*** y una serie de elementos en cantidad indefinida.

El primer elemento (obligatorio en cualquier plan) debe ser el elemento ***nodo\_inicio*** cuya misión es identificar el primer módulo de lógica que se debe ejecutar en el plan, en el ejemplo del servicio 90X de consulta de notas, el ***nodo\_locucion*** “Bienvenida”.

Otros objetos en cantidad indeterminada (cero o más) que puede incorporar cualquier plan de abonado y que no se representan como nodos gráficos enlazados por transiciones en el grafo del plan de abonado, sino que definen propiedades generales del plan, al margen del grafo, son los elementos del tipo: ***definir\_variable***, ***definirBaseDatos*** y ***definir\_recursos***.

El resto de elementos que puede incluir un Plan de Abonado VoiceXML cuyos nombres comienzan por “***nodo\_***” representan los “módulos de lógicas” que tienen representación gráfica en el grafo del plan.

Además de los elementos mencionados, en los nodos que representan una interacción con el usuario para obtener un dato de éste (*nodo\_menu*, *nodo\_dato\_entrada* y *nodo\_grabar\_mensajeVoz*) pueden ocurrir eventos que pueden dar lugar a otras transiciones dependiendo del evento que ocurra en cada momento.

En definitiva, el ciclo de operaciones para el ejemplo planteado se podría resumir en los siguientes puntos:

- La Universidad solicita al operador la contratación de un 90X sobre el que gestionar la consulta de notas por parte de los alumnos.
- El operador desarrolla mediante la herramienta AFABLE IDE de forma gráfica el plan de abonado a medida de lo que solicita la Universidad, pudiendo llevar, incluso “en mano” dicho diseño al cliente.
- Los operadores de red configuran los nodos de Red Inteligente para la asignación de un nuevo 90X que se utilizará con el servicio AFABLE desarrollado.
- Mediante el AFABLE MANAGER, los operadores de red se encargan de realizar el despliegue del plan para el 90X contratado, de forma que el servicio ya esté operativo para la Universidad en un rango de teléfonos de prueba.



- Las pruebas permiten la validación del servicio desplegado, tras lo cual, los operadores activarán el servicio en el AFABLE MANAGER

Una vez implantado el servicio, el AFABLE SERVER se encarga de gestionar las llamadas del usuario final solicitando su nota y de interactuar con todo el entramado de Red Inteligente para llevar a cabo el servicio.

- Por último, el titular del número 90X, en este caso la Universidad, puede mediante el USER WEB a través de una interfaz web gestionar elementos concretos del servicio, consultar informes, rellenar y borrar datos de las tablas, etc.



---

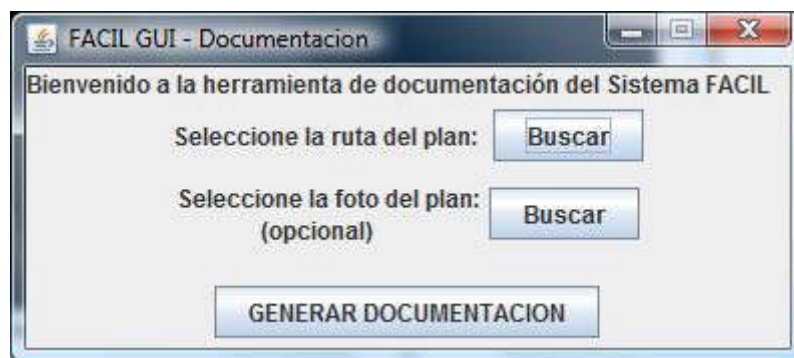
## B MANUAL DE USUARIO

### B.1 ENTRADAS DE LA APLICACIÓN

La aplicación podrá recibir dos parámetros de entrada. El primer parámetro, obligatorio, será el fichero del Plan de Abonado. Dicho fichero, tendrá como extensión *.axml* y deberá ser un fichero válido según la sintaxis del “*XML Schema PlanAbonadoVoiceXML V1.0*” definido y caracterizado en un fichero *.xsd*. Dicho plan se generará y validará con la herramienta gráfica AFABLE IDE, asegurándonos de que el resultado final, es un fichero con sintaxis xml representativo del Servicio 90X contratado por el cliente.

El segundo parámetro, opcional, será un fichero de extensión *.png* ó *.jpg*, con la imagen del Plan de Abonado. Dicha imagen será la representación gráfica que forman los nodos y todas sus transiciones en el AFABLE IDE, aunque deberá conseguirse con un programa de diseño gráfico externo, en base a unas características de formato determinadas. Este segundo parámetro es opcional, con lo que si no se incluye, no se genera en la documentación el apartado específico dedicado a la imagen del Plan de Abonado. A continuación se muestra la pantalla de inicio de la herramienta:

**Figura B-1: Pantalla de Bienvenida del Sistema de Documentación de AFABLE**

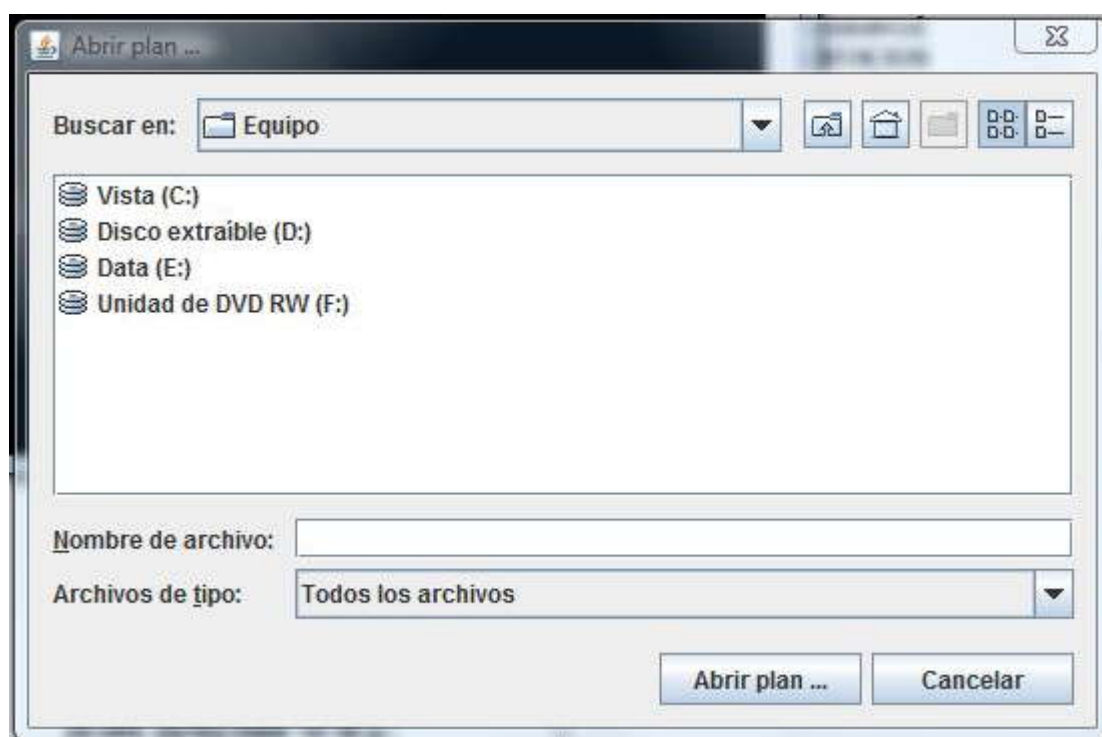


**Parámetros de entrada:**

- Fichero del plan de abonado (.xml): Obligatorio
- Fichero con la imagen del plan (.jpg ó .png): Opcional

Mediante los botones de **“Buscar”**, seleccionamos de nuestro PC los ficheros requeridos, como se muestra en la siguiente imagen:

**Figura B-2: Pantalla de Selección de Ficheros**



Una vez seleccionados los parámetros de entrada pulsamos sobre el botón **“Generar documentación”**, para empezar el proceso de generación automática de documentación del Plan de Abonado.

## B.2 INTERFAZ GRÁFICA DE USUARIO

Una vez que el usuario pulsa sobre el botón de **“Generar documentación”**, se presenta una pequeña interfaz gráfica con un formulario. De este modo, el usuario

puede elegir una serie de opciones a la hora de generar el documento de su Plan de Abonado. En la siguiente imagen se recoge la forma de la interfaz gráfica:

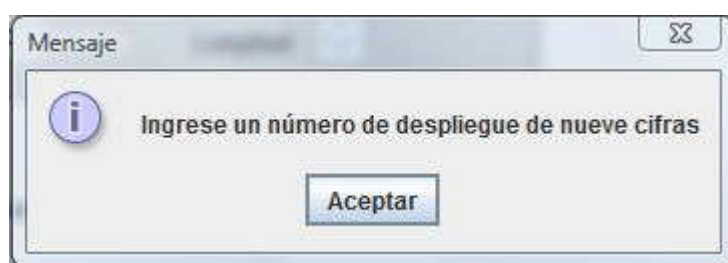
**Figura B-3: Interfaz Gráfica de Usuario**

En la interfaz gráfica de usuario, se pueden distinguir los siguientes puntos:

- **Información de los itinerarios del plan:** En la zona superior de la interfaz gráfica, se muestra información del número de itinerarios de cada tipo que componen el Plan de Abonado. Se puede ver la información del número de itinerarios correctos que hay en el plan, número de itinerarios sin salida válida que existen en el plan, número de itinerarios con errores y número de itinerarios que contienen algún evento de “*marcha atrás*” en el plan.

- **Número de despliegue** (obligatorio): Es el número 90X en el que se va a realizar el despliegue del Plan de Abonado. Tiene, simplemente, carácter informativo y aparecerá en la documentación junto con otros datos generales del Plan de Abonado. Se comprobará, antes de seguir adelante, que dicho número de despliegue es válido, esto es, consta de nueve dígitos. Cualquier error en la inclusión del número de despliegue será informado al usuario, no prosiguiendo la aplicación hasta que no quede correctamente validado. Un ejemplo de aviso de error es el siguiente:

**Figura B-4: Número de despliegue no válido**



- **Tipo de ordenación** (obligatorio): El tipo de ordenación permite establecer el orden en el que queremos que aparezcan los distintos itinerarios dentro de cada uno de los cuatro grupos en los que pueden ubicarse:
  - *Importancia*: Cuando ordenamos los itinerarios por importancia, se analiza el peso específico que tiene cada nodo dentro de su itinerario. Dicho peso, como ya vimos en apartados anteriores, viene determinado por un par de atributos comunes a todos los nodos. Un usuario puede crear itinerarios de mayor o menor importancia reasignando los valores de estos atributos en cada uno de los nodos, de forma que pueda establecer un orden concreto con el que generar un tipo de documentación determinada.
  - *Longitud*: Cuando se ordenan los itinerarios atendiendo a criterios de longitud, se tiene en cuenta, simplemente, el número de nodos que forman parte de cada itinerario, estableciendo un orden, de mayor a

menor, según el número de nodos que puede cursar una llamada dentro de un itinerario concreto del Plan de Abonado.

El tipo de ordenación es obligatorio y excluyente, de forma que sólo se puede elegir un tipo de ordenación de las dos. Si el usuario no selecciona un tipo de ordenación determinada la aplicación le muestra el siguiente mensaje de aviso:

**Figura B-5: Tipo de ordenación inexistente**



- **Ver todos los itinerarios del plan:** (opcional). El usuario puede seleccionar esta opción si lo que desea es incluir en la documentación todos los itinerarios del plan. Esta opción deshabilita todas las demás, como se puede comprobar en la siguiente figura:

**Figura B-6: Ver todos los itinerarios del Plan de Abonado**

The screenshot shows a software window titled "Sistema Documentacion FACIL - Interfaz gráfica de usuario". It contains several input fields and checkboxes for configuring how itineraries are displayed. At the top, four statistics are shown: "Número de itinerarios correctos:" (2), "Número de itinerarios sin salida válida:" (2), "Número de itinerarios con errores:" (3), and "Número de itinerarios con eventos de marcha atrás:" (2). Below these is a "Número de despliegue:" field with the value "900100300" and a checked checkbox "Ver todos los itinerarios del plan". A "Tipo de ordenacion" section has two radio buttons: "Importancia" (unselected) and "Longitud" (selected). Below this are five rows of options, each with a checkbox and a text field: "Ver el itinerario de mayor importancia / longitud" (checkbox), "Ver todos los itinerarios correctos del plan" (checkbox, text field "Ver sólo algunos ..."), "Ver todos los itinerarios sin salida válida del plan" (checkbox, text field "Ver sólo algunos ..."), "Ver todos los itinerarios con errores del plan" (checkbox, text field "Ver sólo algunos ..."), and "Ver todos los itinerarios con eventos de marcha atrás en el plan" (checkbox, text field "Ver sólo algunos ..."). An "Aceptar" button is at the bottom right.

Estadística	Valor
Número de itinerarios correctos:	2
Número de itinerarios sin salida válida:	2
Número de itinerarios con errores:	3
Número de itinerarios con eventos de marcha atrás:	2

Número de despliegue: 900100300 Ver todos los itinerarios del plan ☒

Tipo de ordenacion

Importancia ☐ Longitud ☒

Ver el itinerario de mayor importancia / longitud ☐

Ver todos los itinerarios correctos del plan ☐ Ver sólo algunos ...

Ver todos los itinerarios sin salida válida del plan ☐ Ver sólo algunos ...

Ver todos los itinerarios con errores del plan ☐ Ver sólo algunos ...

Ver todos los itinerarios con eventos de marcha atrás en el plan ☐ Ver sólo algunos ...

Aceptar

- **Ver el itinerario de mayor importancia / longitud** (opcional): También denominado itinerario principal. Se entiende por itinerario principal del Plan de Abonado al primer itinerario correcto ordenado según la opción escogida por el usuario en el punto anterior. Cuando se selecciona esta opción se incluye en la documentación una sección independiente con la descripción de todos los nodos que forman este itinerario principal.
- **Ver todos los itinerarios correctos** (opcional): Si el usuario marca esta opción, se incluye en la documentación una sección con la información de todos los itinerarios correctos del Plan de Abonado. Por itinerario correcto, se entiende, la secuencia de nodos por los que pasa una llamada, de modo

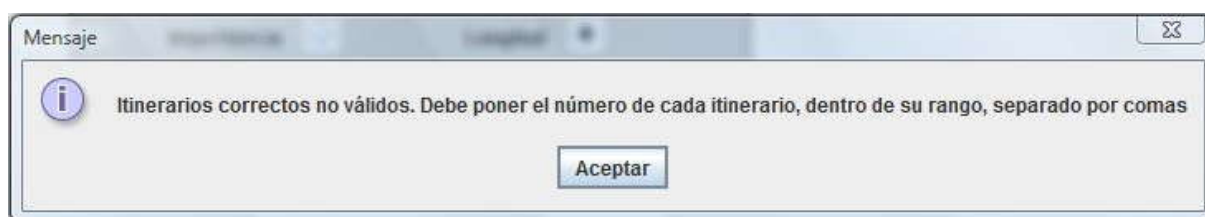


que todas las transiciones producidas entre ellos son correctas desde el punto de vista de la lógica del Servicio 90X. Las transiciones correctas, se representan en el AFABLE IDE con flechas de color negro.

- **Ver todos los itinerarios sin salida válida** (opcional): Si el usuario marca esta opción, se incluye en la documentación una sección con la información de todos los itinerarios sin salida válida del Plan de Abonado. Por itinerarios sin salida válida se entienden todos aquellos caminos en el Plan de Abonado en los que existe al menos un nodo en el que se produce una transición concreta por la que la llamada se cursa por un nodo alternativo. Dichas transiciones se representan en el AFABLE IDE con flechas de color azul. Los eventos que producen transiciones de este tipo son los siguientes:
  - Transición de tipo “*no-input*”. Se producen cuando la lógica del Servicio 90X espera un dato del llamante y éste no introduce ninguno, bien sea porque permanece callado, bien porque no pulsa ninguna tecla. En este caso, se espera a que finalice un temporizador antes de lanzarse el evento.
  - Transición de tipo “*no-match*”. Se producen cuando la lógica del Servicio 90X espera un dato del llamante y éste introduce uno que el sistema no puede reconocer. Se establece un número máximo de intentos, cubiertos los cuáles, se dispara el evento.
- **Ver todos los itinerarios con errores** (opcional): Se entiende por itinerarios con errores a aquéllos en los que existe al menos un nodo en el que puede producirse una situación anómala, no controlada por la lógica del Servicio 90X, por la que la llamada debe evolucionar por un camino distinto. Se produce un error, por ejemplo, cuando al utilizar un servicio externo se produce un fallo de conexión con la máquina que sirve la petición. Las transiciones con errores se representan en el AFABLE IDE mediante flechas de color rojo.

- **Ver todos los itinerarios con eventos de marcha** atrás (opcional): Se producen cuando en el Plan de Abonado del Servicio 90X, existe al menos un nodo en el que el llamante puede, voluntariamente, hacer que la llamada curse por un nodo anterior en el Plan de Abonado. Dichos eventos tienen un tratamiento especial a la hora de generar la documentación correspondiente, puesto que provocan “bucles” que es necesario romper para poder completar los diferentes itinerarios.
- **Ver sólo algunos ...** (opcional) Para cada uno de los cuatro tipos de itinerarios, es posible seleccionar un subconjunto de caminos a incluir en la documentación. Puede ocurrir que un Plan de Abonado concreto, cuente con un gran número de itinerarios correctos, y sólo se quiera que figure en la documentación la información de unos pocos de ellos. Para ello, es necesario incluir en los cuadros de texto el número de los itinerarios separados por comas. Los itinerarios se validarán en base a dos criterios. El primero, que lo que introduce el usuario sean números, separados por comas y segundo, que dichos números se encuentren en el rango correcto de cada uno de los tipos de itinerarios existentes. Así pues, si existen 2 itinerarios correctos sólo se podrá incluir el itinerario 1, el 2, o ambos: 1, 2. En caso de que la validación no sea correcta, se presenta el siguiente aviso al usuario:

**Figura B-7: Itinerarios individuales no válidos**



La opción de ver sólo algunos itinerarios queda deshabilitada si se selecciona la opción correspondiente de “Ver todos los itinerarios ...”, para cada una de las categorías. En las siguientes figuras, se muestra un ejemplo de cómo rellenar itinerarios concretos para la generación de documentación, así como

también cómo estos campos quedan deshabilitados al elegir la opción más general en cada categoría.

**Figura B-8: Itinerarios individuales correctos**

The screenshot shows a window titled "Sistema Documentacion FACIL - Interfaz gráfica de usuario". It contains the following elements:

- Four statistics at the top, each with a label and a text input field:
  - Número de itinerarios correctos: 2
  - Número de itinerarios sin salida válida: 2
  - Número de itinerarios con errores: 3
  - Número de itinerarios con eventos de marcha atrás: 2
- A horizontal separator line.
- A "Número de despliegue:" label followed by a text input field containing "900100300".
- A "Ver todos los itinerarios del plan" label followed by an unchecked checkbox.
- A "Tipo de ordenacion" section with two radio buttons:
  - "Importancia" (unchecked)
  - "Longitud" (checked)
- A "Ver el itinerario de mayor importancia / longitud" label followed by an unchecked checkbox.
- A list of five filter options, each with an unchecked checkbox and a "Ver sólo algunos ..." label followed by a text input field:
  - Ver todos los itinerarios correctos del plan: 1, 3, 7, 9
  - Ver todos los itinerarios sin salida válida del plan: (empty)
  - Ver todos los itinerarios con errores del plan: (empty)
  - Ver todos los itinerarios con eventos de marcha atrás en el plan: (empty)
- An "Aceptar" button at the bottom center.

**Figura B-9: Itinerarios individuales deshabilitados**

Sistema Documentacion FACIL - Interfaz gráfica de usuario

Número de itinerarios correctos: 2

Número de itinerarios sin salida válida: 2

Número de itinerarios con errores: 3

Número de itinerarios con eventos de marcha atrás: 2

Número de despliegue: 900100300

Ver todos los itinerarios del plan ☐

Tipo de ordenacion

Importancia ☐ Longitud ☒

Ver el itinerario de mayor importancia / longitud ☒

Ver todos los itinerarios correctos del plan ☒ Ver sólo algunos ...

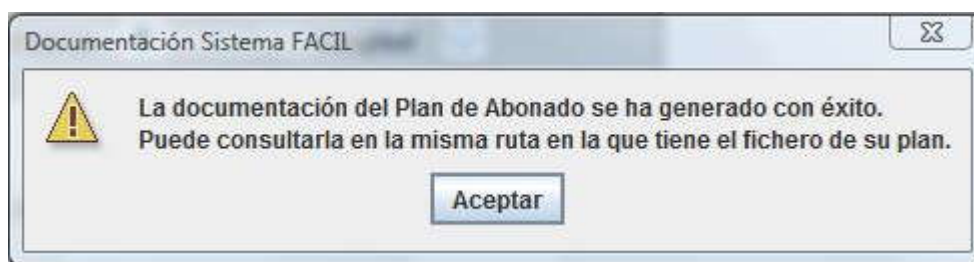
Ver todos los itinerarios sin salida válida del plan ☒ Ver sólo algunos ...

Ver todos los itinerarios con errores del plan ☒ Ver sólo algunos ...

Ver todos los itinerarios con eventos de marcha atrás en el plan ☒ Ver sólo algunos ...

Aceptar

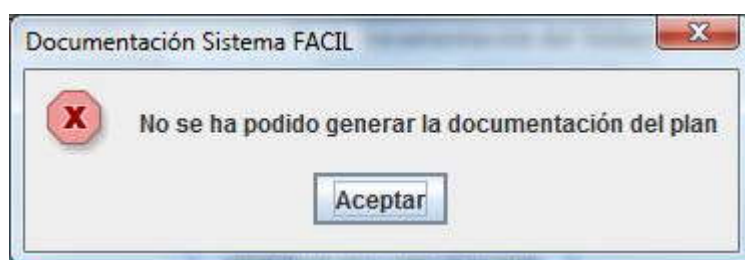
Una vez seleccionadas las opciones deseadas, es necesario pulsar el botón “Aceptar” para generar el documento del Plan de Abonado. La aplicación, en este punto, trata de construir el documento. Si la generación del mismo es correcta, se informa al usuario con el sistema mensaje:

**Figura B-10: Generación correcta de la documentación**

El documento generado, con extensión “.odt”, se guarda en la misma ruta que introdujo el usuario para el fichero del plan y tendrá como nombre el mismo nombre del Plan de Abonado.

Por el contrario, si se produce algún error en la generación de la documentación, la aplicación avisa al usuario con el siguiente mensaje de error:

**Figura B-11: Error en la generación de la documentación**





---

# GLOSARIO DE TÉRMINOS Y ACRÓNIMOS

## **90X**

Identificación genérica de los números de teléfono de acceso a servicios de Red Inteligente

## **AFABLE**

Sistema de Automatización de Funciones de Atención telefónica Basado en Lógicas Especiales. Por cuestiones de mercado, el Sistema AFABLE se comercializa con el nombre de Sistema **FACIL**.

## **API**

Application Programming Interface

## **AXML**

Afable eXtensible Markup Language

## **CAR**

Contestador Automático en Red de Telefónica de España

## **DOM**

Document Object Model

## **DTMF**

Dual-Tone Multi-Frequency

## **ECMAScript**

Especificación de lenguaje de programación publicada por ECMA International. Actualmente está aceptado como el estándar ISO 16262

## **GIF**

Graphics Interchange Format

## **GSL**

Grammar Specification Language

## **HTML**

HyperText Markup Language

## **HTTP**

HyperText Transfer Protocol

## **IBM**

International Business Machines

**IDABC**

Interoperable Delivery of European eGovernment Service to public Administrations, Business and Citizens

**IDE**

Integrated Development Environment

**IVR**

Interactive Voice Response

**JDBC**

Java Database Connectivity API

**JDK**

Java Development Kit

**JPG**

Joint Photographic Experts Group

**JRE**

Java Runtime Environment

**MIME**

Multipurpose Internet Mail Extensions

**ML**

Módulo de Lógica

**MMS**

Multimedia Messaging System

**MP3**

MPEG-1 Audio Layer 3. Es un formato de audio digital comprimido con pérdida desarrollado por el Moving Picture Experts Group (MPEG)

**OASIS**

Advancing Open Standards for the Information Society

**ODF**

OpenDocument Format

**ODFDOM**

OpenDocument Format Document Object Model. API para el tratamiento de ficheros OpenDocument

**OOXML**

Open Office XML



**PC**

Personal Computer

**PDF**

Portable Document Format

**PFC**

Proyecto Fin de Carrera

**PNG**

Portable Network Graphics

**POO**

Programación Orientada a Objetos

**RI**

Red Inteligente

**SMS**

Short Message Service

**SQL**

Structured Query Language

**UML**

Unified Modeling Language

**VoiceXML**

Voice Extensible Markup Language. Lenguaje diseñado para crear aplicaciones interactivas basadas en diálogos de voz (IVR - Interactive Voice Response). El objetivo es llevar las ventajas de las tecnologías Web a las aplicaciones de respuesta interactiva de voz

**VOX**

Formato de audio "Dialogic ADPCM". Es un formato optimizado para la voz humana y ampliamente utilizado en aplicaciones de telefonía

**WAV**

WAVEform audio format, es un formato de audio digital normalmente sin compresión de datos desarrollado y propiedad de Microsoft

**XML**

eXtensible Markup Language

**XPath**

XML Path Language

**XSD**

XML Schema Definition

---

# BIBLIOGRAFÍA

- [1] **Arquitectura de AFABLE v1.0. Edición: 00.00 Rev: 07 22/05/2008**  
División de Servicios Convergentes de Valor Añadido  
Telefónica I + D
- [2] **Manual de Usuario del Sistema AFABLE v1.0. Edición: 02.00 Rev: 01 04/12/2008**  
División de Servicios Convergentes de Valor Añadido  
Telefónica I + D
- [3] **Formato de Documento Abierto para Aplicaciones Ofimáticas (OpenDocument) v1.1**  
Estándar OASIS, 1 Febrero 2007. Michael Brauer, Sun Microsystems, Inc.  
URIs de la especificación:  
  
<http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1.odt>  
<http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1.pdf>
- [4] **OASIS OpenDocument Essentials**  
Using OASIS OpenDocument XML  
J. David Eisenberg
- [5] **Extensible Stylesheet Language (XSL) Version 1.0**  
W3C Recommendation 15 October 2002.  
URIs de la especificación:  
  
[http://www.w3.org/TR/2001/REC-xsl-20011015/\(PDF by RenderX, XML file, HTML \(one large file\), ZIP file\)](http://www.w3.org/TR/2001/REC-xsl-20011015/(PDF%20by%20RenderX,%20XML%20file,%20HTML%20(one%20large%20file),%20ZIP%20file))
- [6] **OpenDocument API (ODFDOM)**  
**ODF Toolkit.**  
<http://odftoolkit.org/projects/odfdom/pages/Home>  
  
Vídeo de presentación del API ODFDOM. 5 Enero de 2009.  
<http://video.google.com/videoplay?docid=-1551869922315985463>  
Svante Schubert

- [7] **Manual OpenOffice**  
<http://www.manualespdf.es/manual-openoffice/>
- [8] **Mapeo de XML a Java. SAX (Simple Api for XML)**  
Alberto Molpeceres, socio Fundador de *javaHispano*  
<http://www.javahispano.org/contenidos/archivo/2/sax1.pdf>
- [9] **“The Complete Manual log4j”**. The reliable, fast and flexible logging framework for Java  
Ceki Gülkü foreword by Scott Stark  
  
Otras URIs relacionadas con Log4j:  
<http://es.wikipedia.org/wiki/Log4j>  
<http://logging.apache.org/log4j/1.2/apidocs/index.html>
- [10] **“Trabajando con GUI en Java Componente java.swing”**  
Wilder López Meléndez  
26 de Junio de 2007  
<http://espanol.geocities.com/wlopezm/articulos.html>
- [11] **Manual Microsoft Project 2003**  
<http://www.aulafacil.com/microsoftproject/curso/Temario.htm>
- [12] **Conceptos sobre estándares abiertos. Comparativa estándares abiertos vs estándares cerrados**  
<http://osluz.unizar.es/proyectos/estandares/definicion>  
<http://people.ffii.org/~abarrio/estandares/Definiciones-oficiales-Estandares-abiertos-20070409a.pdf>  
[http://osluz.unizar.es/files/presentacion\\_estandares.pdf](http://osluz.unizar.es/files/presentacion_estandares.pdf)
- [13] **Documentación Pluggin eUML2 para Eclipse**  
<http://www.soyatec.com/euml2/documentation/com.soyatec.euml2.doc/>
- [14] **Tutorial de XPath**  
Versión: 1.0 Enero 2001

Víctor Manuel Rivas Santos

<http://kal-el.ugr.es/~victor/cursillos/xml/XPath/>

**[15] Apache Ant 1.7.1 Manual**

<http://ant.apache.org/manual/index.html>

**[16] José A. Mañas. “Prueba de Programas”**

<http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm>